WE ARE THE CHAMPIONS

PREDICTIVE METHODS OF DATA MINING
JUNE 2023

Group 7

Carolina Santos (20220089)

Guilherme Figueiredo (20220533)

Ricardo Fangueiro (20220536)

Rodrigo Freire (20221292)

Valeria Caraus (20220316)

Abstract

This work aims to achieve a structured and logical output by applying statistical techniques, crucial for this analysis, such as predictive models, machine learning and more. These models belong to the group of Supervised Learning, where the main idea is to “extract information from a data set and transform it into an understandable structure for future use”, relying on training, and where we input tagged data samples.

Translating the capacity of a prediction model to predict future outcomes, the idea is to take data into account, learn from it, transform it into information and, furthermore, into knowledge. The importance of this study is related with the capacity to retrieve value from data. The result we tried to achieve was being able to create a model capable of taking new data, after training and testing, and retrieve valuable outputs.

For the practical work, as also for the structure of the present document, we applied the methodology (CRISP-DM) and researched to correspond to the three main components of a machine learning algorithm: Representation, Evaluation and Optimization.

Index

1. Introduction

We Are the Champions (W.A.C) is a renowned and trusted sports technology company with extensive experience in data collection and analysis. The organization has realized the potential of **machine learning** to revolutionize the way they predict the outcome of a competition, and our group was chosen to build a model that provides an accurate prediction of an athlete's outcome.

The dataset used contains information such as the athlete's age, previous competition scores, education level, income and various types of training sessions conducted.

The data was collected from various sources around the world, including professional athletic organizations, sports archives, and publicly available performance statistics.

The goal of this machine learning project is to build a predictive model that uses this dataset of professional athletes to determine if they will win or lose a competition. Additionally, this report describes the analytical processes and the conclusions obtained by our group.

2. Methodology

The approach the group decided to follow was the **CRISP-DM** methodology, which is widely known for conducting data mining projects.

The Cross Industry Standard Process for Data Mining consists of six phases, the first being **Business Understanding**, where the objectives and requirements of the project are defined. This step was already conducted in the introduction of the report.

The second phase is **Data Understanding**, which is when we start exploring the available data, by getting our initial insights and checking the quality of the data.

The next phase is **Data Preparation**, which is one of the most important steps. It is where we prepare the data for modelling, which means selecting the data, cleaning it, building it, integrating it, and formatting it.

After preparing the data, the next phase is **Modelling**, where we apply various modelling techniques.

The fifth phase is **Evaluation**, where we assess the results, review the process, and select the model that has the highest quality and best fits our business needs.

Finally, the last phase is **Deployment**, where the chosen model is integrated.

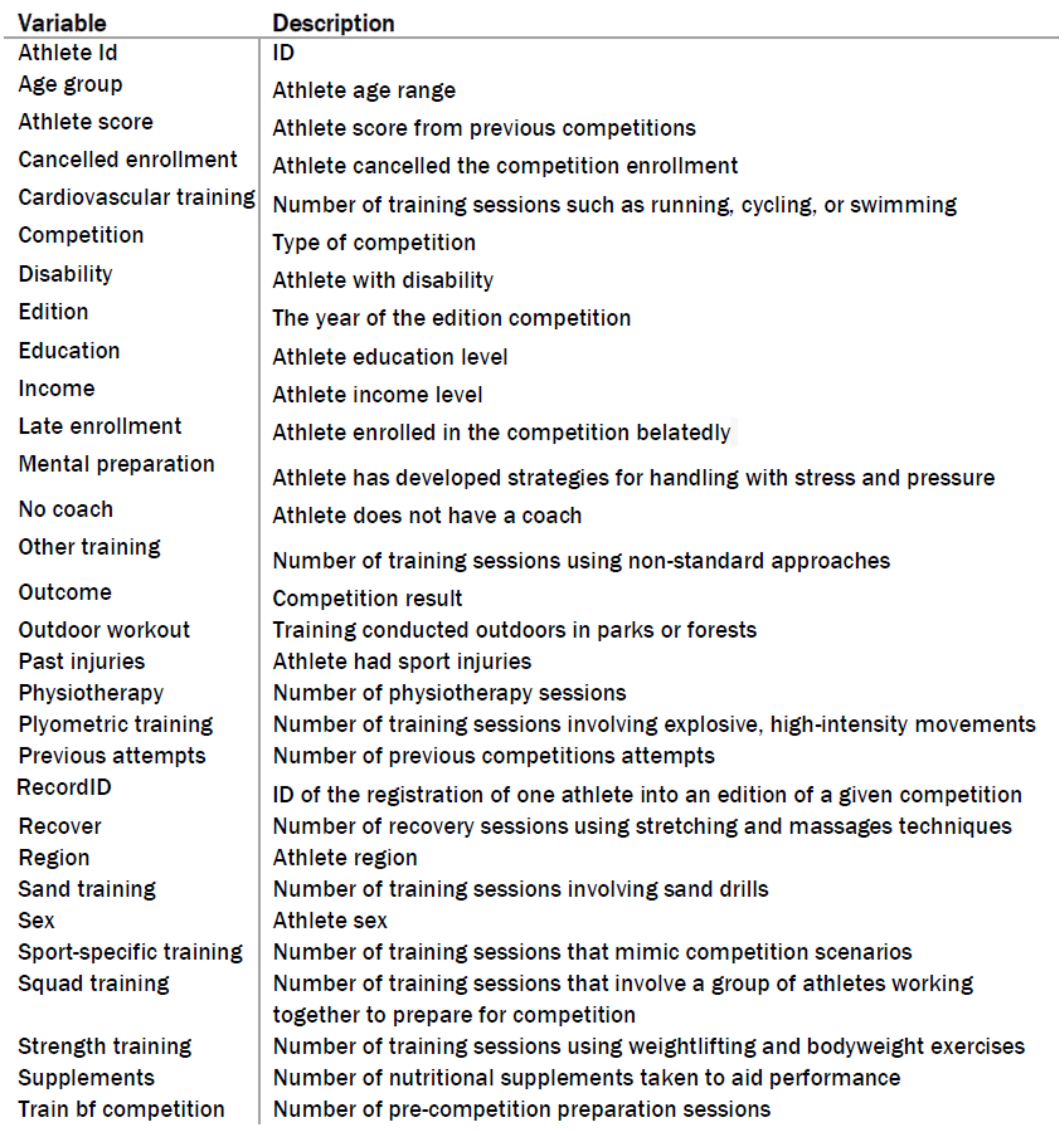
It is important to mention that despite the sequential phases in CRIPS-DM, it is common to go back and forth when necessary.

3. Data Exploration and Understanding

The first step involves **importing** and integrating the files that will be used as a basis for the development of the project - "test" and "train" - both in **CSV format**.

3.1. Variables Description

Below, a list of the **variables** incorporated in both files mentioned above is presented, as well as a brief description of them.

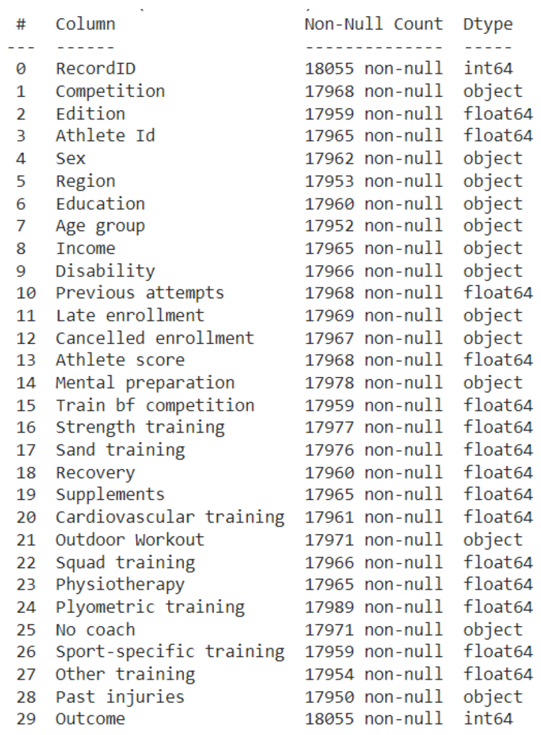


Variable	Description
Athlete Id	ID
Age group	Athlete age range
Athlete score	Athlete score from previous competitions
Cancelled enrollment	Athlete cancelled the competition enrollment
Cardiovascular training	Number of training sessions such as running, cycling, or swimming
Competition	Type of competition
Disability	Athlete with disability
Edition	The year of the edition competition
Education	Athlete education level
Income	Athlete income level
Late enrollment	Athlete enrolled in the competition belatedly
Mental preparation	Athlete has developed strategies for handling with stress and pressure
No coach	Athlete does not have a coach
Other training	Number of training sessions using non-standard approaches
Outcome	Competition result
Outdoor workout	Training conducted outdoors in parks or forests
Past injuries	Athlete had sport injuries
Physiotherapy	Number of physiotherapy sessions
Plyometric training	Number of training sessions involving explosive, high-intensity movements
Previous attempts	Number of previous competitions attempts
RecordID	ID of the registration of one athlete into an edition of a given competition
Recover	Number of recovery sessions using stretching and massages techniques
Region	Athlete region
Sand training	Number of training sessions involving sand drills
Sex	Athlete sex
Sport-specific training	Number of training sessions that mimic competition scenarios
Squad training	Number of training sessions that involve a group of athletes working together to prepare for competition
Strength training	Number of training sessions using weightlifting and bodyweight exercises
Supplements	Number of nutritional supplements taken to aid performance
Train bf competition	Number of pre-competition preparation sessions

Table 1 Variables Description

3.2. Statistical Details

After obtaining information about each of the variables under analysis, and realizing what their meaning and what they represent, it was important to identify their **type of variable** and if there were **missing values**, which would need treatment at a later stage.

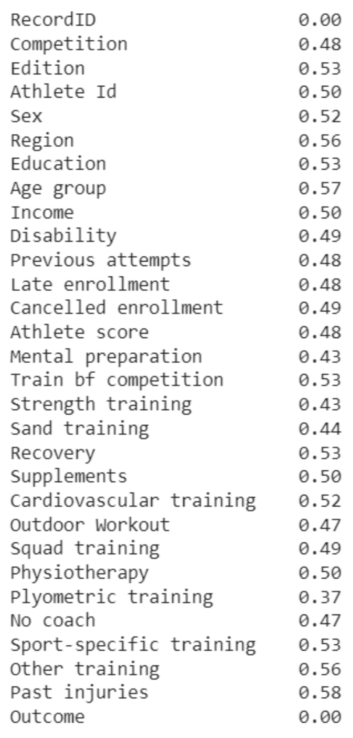


```
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   RecordID                18055 non-null  int64
 1   Competition             17968 non-null  object
 2   Edition                 17959 non-null  float64
 3   Athlete Id              17965 non-null  float64
 4   Sex                     17962 non-null  object
 5   Region                  17953 non-null  object
 6   Education               17960 non-null  object
 7   Age group               17952 non-null  object
 8   Income                  17965 non-null  object
 9   Disability              17966 non-null  object
 10  Previous attempts       17968 non-null  float64
 11  Late enrollment         17969 non-null  object
 12  Cancelled enrollment    17967 non-null  object
 13  Athlete score           17968 non-null  float64
 14  Mental preparation      17978 non-null  object
 15  Train bf competition    17959 non-null  float64
 16  Strength training       17977 non-null  float64
 17  Sand training           17976 non-null  float64
 18  Recovery                17960 non-null  float64
 19  Supplements             17965 non-null  float64
 20  Cardiovascular training 17961 non-null  float64
 21  Outdoor Workout         17971 non-null  object
 22  Squad training          17966 non-null  float64
 23  Physiotherapy           17965 non-null  float64
 24  Plyometric training     17989 non-null  float64
 25  No coach                17971 non-null  object
 26  Sport-specific training 17959 non-null  float64
 27  Other training          17954 non-null  float64
 28  Past injuries           17950 non-null  object
 29  Outcome                 18055 non-null  int64
```

Table 2 Variables Data Types

As mentioned in the table above, the train dataset had variables of three types:

- 15 variables with the data type "**float**";
- 2 variables with the data type "**integer**";
- 13 variables with the data type "**object**", which match the categorical variables.



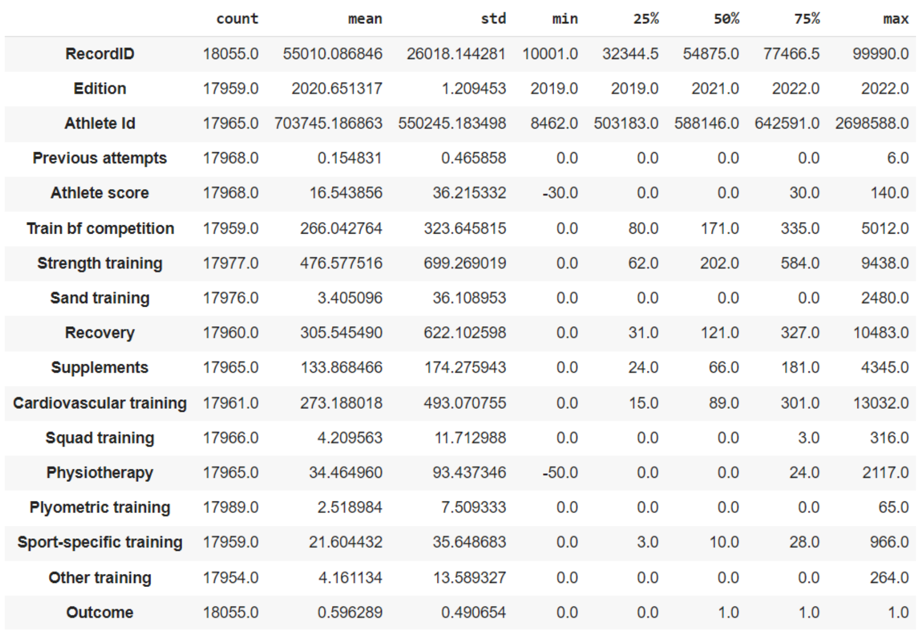
```
RecordID                  0.00
Competition               0.48
Edition                   0.53
Athlete Id                0.50
Sex                       0.52
Region                    0.56
Education                 0.53
Age group                 0.57
Income                    0.50
Disability                0.49
Previous attempts         0.48
Late enrollment           0.48
Cancelled enrollment      0.49
Athlete score             0.48
Mental preparation        0.43
Train bf competition      0.53
Strength training         0.43
Sand training             0.44
Recovery                  0.53
Supplements               0.50
Cardiovascular training   0.52
Outdoor Workout           0.47
Squad training            0.49
Physiotherapy             0.50
Plyometric training       0.37
No coach                  0.47
Sport-specific training   0.53
Other training            0.56
Past injuries             0.58
Outcome                   0.00
```

Table 3 Variables Percentage of Missing Values

In addition, the **missing values** of each variable were also checked, where it was possible to determine that, except for the variable "RecordID" and the target variable ("Outcome"), all variables contained a percentage of missing values, albeit quite residual. For this reason, the most appropriate techniques for filling in missing values will be put into practice later.

3.3. Descriptive Statistics | Numerical Features

Descriptive statistics are mostly used to summarize the numerical features contained in our dataset, which helped us to detect anomalies and grasp some main insights. The *describe* method builds a table, similar to the one presented below, where the most important statistics for each variable are listed (i.e., the count, mean, standard deviation, minimum and maximum value, and also the quantiles).

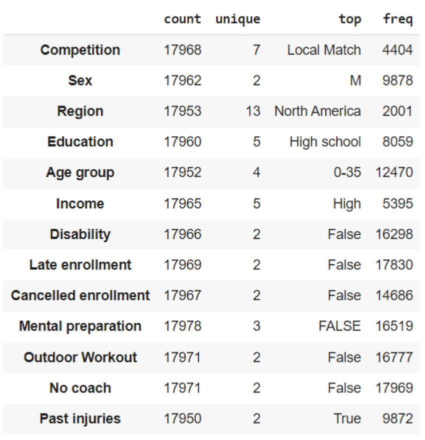


	count	mean	std	min	25%	50%	75%	max
RecordID	18055.0	55010.086846	26018.144281	10001.0	32344.5	54875.0	77466.5	99990.0
Edition	17959.0	2020.651317	1.209453	2019.0	2019.0	2021.0	2022.0	2022.0
Athlete Id	17965.0	703745.186863	550245.183498	8462.0	503183.0	588146.0	642591.0	2698588.0
Previous attempts	17968.0	0.154831	0.465858	0.0	0.0	0.0	0.0	6.0
Athlete score	17968.0	16.543856	36.215332	-30.0	0.0	0.0	30.0	140.0
Train bf competition	17959.0	266.042764	323.645815	0.0	80.0	171.0	335.0	5012.0
Strength training	17977.0	476.577516	699.269019	0.0	62.0	202.0	584.0	9438.0
Sand training	17976.0	3.405096	36.108953	0.0	0.0	0.0	0.0	2480.0
Recovery	17960.0	305.545490	622.102598	0.0	31.0	121.0	327.0	10483.0
Supplements	17965.0	133.868466	174.275943	0.0	24.0	66.0	181.0	4345.0
Cardiovascular training	17961.0	273.188018	493.070755	0.0	15.0	89.0	301.0	13032.0
Squad training	17966.0	4.209563	11.712988	0.0	0.0	0.0	3.0	316.0
Physiotherapy	17965.0	34.464960	93.437346	-50.0	0.0	0.0	24.0	2117.0
Plyometric training	17989.0	2.518984	7.509333	0.0	0.0	0.0	0.0	65.0
Sport-specific training	17959.0	21.604432	35.648683	0.0	3.0	10.0	28.0	966.0
Other training	17954.0	4.161134	13.589327	0.0	0.0	0.0	0.0	264.0
Outcome	18055.0	0.596289	0.490654	0.0	0.0	1.0	1.0	1.0

Table 4 Descriptive Statistics for Numerical Data

3.4. Summary Statistics | Categorical Features

In turn, it is also necessary to carry out a similar analysis, but this time for **categorical variables**, where the statistics used are the count of each variable, the number of unique values, the most frequent class, and the respective frequency number.

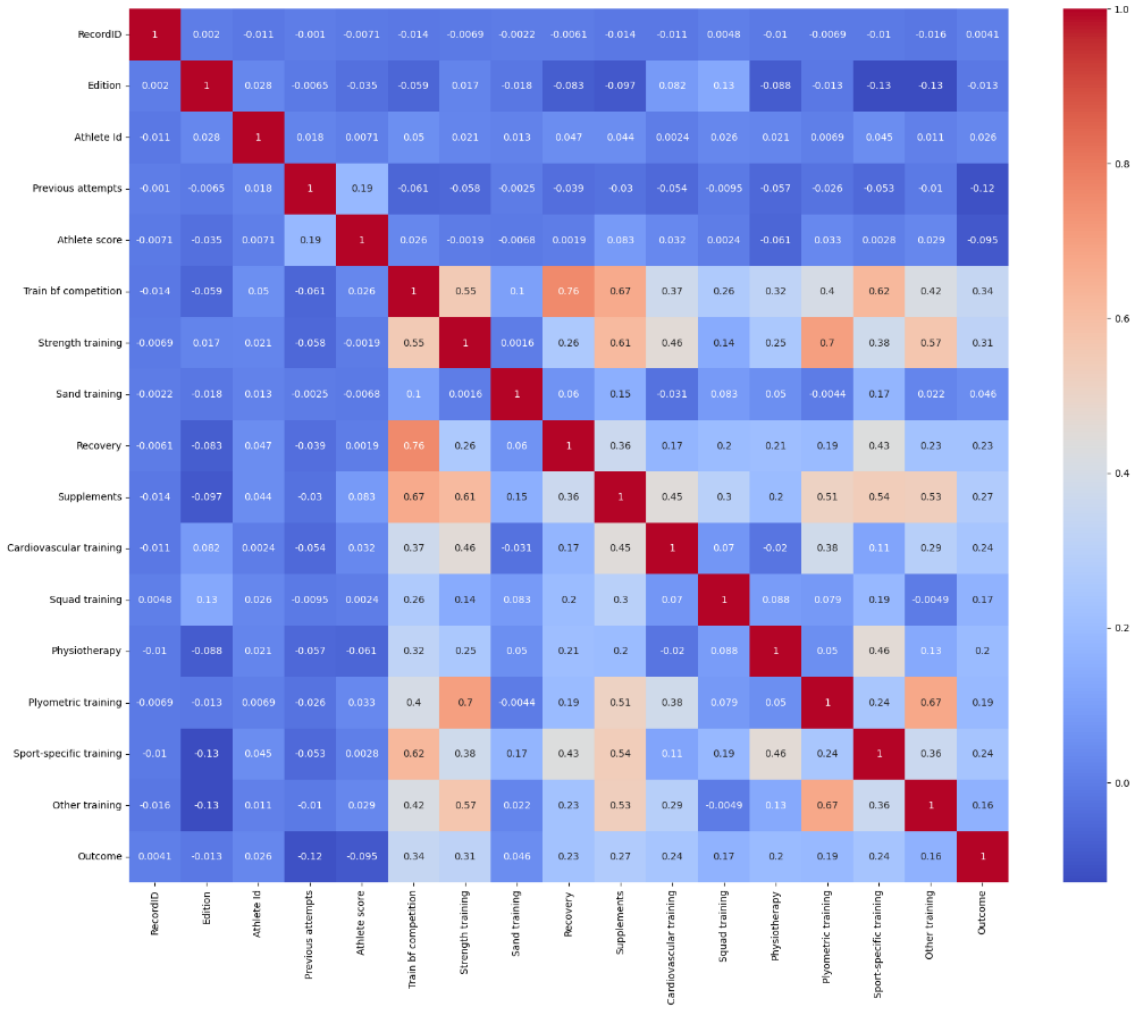


	count	unique	top	freq
Competition	17968	7	Local Match	4404
Sex	17962	2	M	9878
Region	17953	13	North America	2001
Education	17960	5	High school	8059
Age group	17952	4	0-35	12470
Income	17965	5	High	5395
Disability	17966	2	False	16298
Late enrollment	17969	2	False	17830
Cancelled enrollment	17967	2	False	14686
Mental preparation	17978	3	FALSE	16519
Outdoor Workout	17971	2	False	16777
No coach	17971	2	False	17969
Past injuries	17950	2	True	9872

Table 5 Summary Statistics for Categorical Data

3.5. Correlations

Correlation is a very useful measure that allows us to identify the **relationships** between the different variables in our dataset. However, it does not imply causation, which means we must be careful and be able to identify which ones actually have informational relationships. To do so, we created a **Pearson Correlation Matrix** to have a summary of the linear relation between numerical features.



	RecordID	Edition	Athlete Id	Previous attempts	Athlete score	Train bf competition	Strength training	Sand training	Recovery	Supplements	Cardiovascular training	Squad training	Physiotherapy	Plyometric training	Sport-specific training	Other training	Outcome
RecordID	1	0.002	-0.011	-0.001	-0.0071	-0.014	-0.0069	-0.0022	-0.0061	-0.014	-0.011	0.0048	-0.01	-0.0069	-0.01	-0.016	0.0041
Edition	0.002	1	0.028	-0.0065	-0.035	-0.059	0.017	-0.018	-0.083	-0.097	0.082	0.13	-0.088	-0.013	-0.13	-0.13	-0.013
Athlete Id	-0.011	0.028	1	0.018	0.0071	0.05	0.021	0.013	0.047	0.044	0.0024	0.026	0.021	0.0069	0.045	0.011	0.026
Previous attempts	-0.001	-0.0065	0.018	1	0.19	-0.061	-0.058	-0.0025	-0.039	-0.03	-0.054	-0.0095	-0.057	-0.026	-0.053	-0.01	-0.12
Athlete score	-0.0071	-0.035	0.0071	0.19	1	0.026	-0.0019	-0.0068	0.0019	0.083	0.032	0.0024	-0.061	0.033	0.0028	0.029	-0.095
Train bf competition	-0.014	-0.059	0.05	-0.061	0.026	1	0.55	0.1	0.76	0.67	0.37	0.26	0.32	0.4	0.62	0.42	0.34
Strength training	-0.0069	0.017	0.021	-0.058	-0.0019	0.55	1	0.0016	0.26	0.61	0.46	0.14	0.25	0.7	0.38	0.57	0.31
Sand training	-0.0022	-0.018	0.013	-0.0025	-0.0068	0.1	0.0016	1	0.06	0.15	-0.031	0.083	0.05	-0.0044	0.17	0.022	0.046
Recovery	-0.0061	-0.083	0.047	-0.039	0.0019	0.76	0.26	0.06	1	0.36	0.17	0.2	0.21	0.19	0.43	0.23	0.23
Supplements	-0.014	-0.097	0.044	-0.03	0.083	0.67	0.61	0.15	0.36	1	0.45	0.3	0.2	0.51	0.54	0.53	0.27
Cardiovascular training	-0.011	0.082	0.0024	-0.054	0.032	0.37	0.46	-0.031	0.17	0.45	1	0.07	-0.02	0.38	0.11	0.29	0.24
Squad training	0.0048	0.13	0.026	-0.0095	0.0024	0.26	0.14	0.083	0.2	0.3	0.07	1	0.088	0.079	0.19	-0.0049	0.17
Physiotherapy	-0.01	-0.088	0.021	-0.057	-0.061	0.32	0.25	0.05	0.21	0.2	-0.02	0.088	1	0.05	0.46	0.13	0.2
Plyometric training	-0.0069	-0.013	0.0069	-0.026	0.033	0.4	0.7	-0.0044	0.19	0.51	0.38	0.079	0.05	1	0.24	0.67	0.19
Sport-specific training	-0.01	-0.13	0.045	-0.053	0.0028	0.62	0.38	0.17	0.43	0.54	0.11	0.19	0.46	0.24	1	0.36	0.24
Other training	-0.016	-0.13	0.011	-0.01	0.029	0.42	0.57	0.022	0.23	0.53	0.29	-0.0049	0.13	0.67	0.36	1	0.16
Outcome	0.0041	-0.013	0.026	-0.12	-0.095	0.34	0.31	0.046	0.23	0.27	0.24	0.17	0.2	0.19	0.24	0.16	1

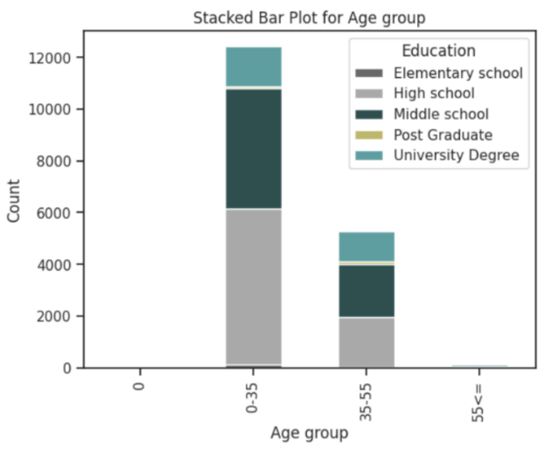
Figure 1 Heatmap of Pearson Correlations

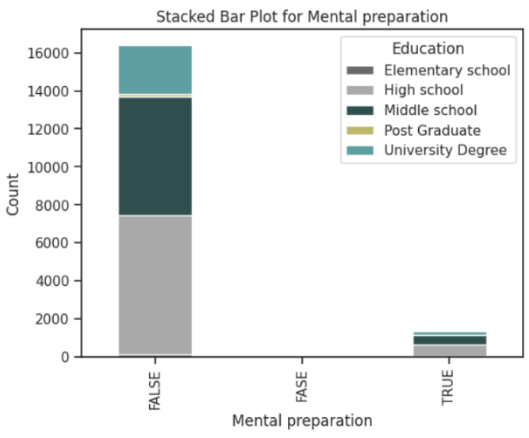
3.6. Coherence Checking

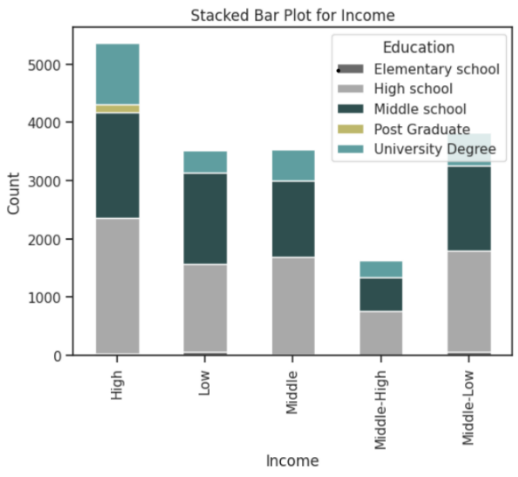
This was the phase where we tried to detect **inconsistencies** in our dataset. This analysis will help us to have a cleaner model capable of making more accurate predictions for unseen data. Therefore, in addition to observing and analyzing the statistics for each variable, we also used different visualizations, namely:

- **Scatter Plots**: to perceive the relationship and behavior between some of the numerical variables;
- **Count Plots**: to understand the distribution of the *Sex* variable for each of the existing regions in the dataset and if there were regions that did not make sense to exist, or that, on the other hand, could be grouped to another class;
- **Bar Plots**: that were used to understand which classes have the highest occurrence for each of the categorical variables and if there are inconsistent classes;
- **Scatter Matrix Plot**: that allowed to perceive the distribution of the values for each one of the numerical variables;
- **Stacked Bar Plots**: that allow us to understand the frequency of the variable *Education* in the remaining categorical variables.

As an example, some of the Stacked Bar Plots obtained are presented below, where it was possible to detect some inconsistencies or classes that could be grouped together, which will be dealt with later in the data preprocessing phase.







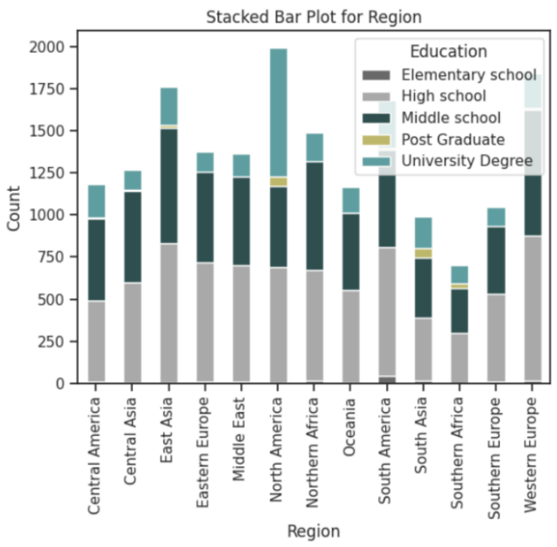


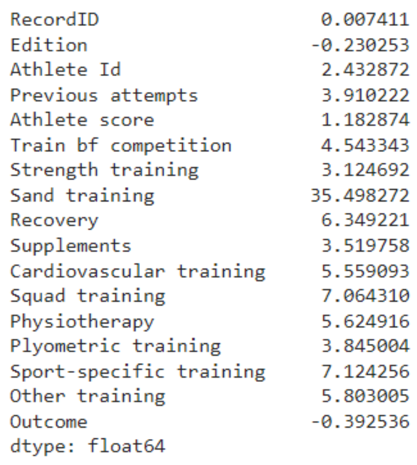
Figure 2 Stacked Bar Plots for Incoherence detection

3.7. Outliers

Another aspect that can influence the performance of our model are the **outliers**. To avoid bias caused by any records with abnormal values, we resorted to boxplots that would allow us to identify them. In section 5.2, it is possible to see what was done to overcome the effects of outliers detected through boxplots.

3.8. Skewness

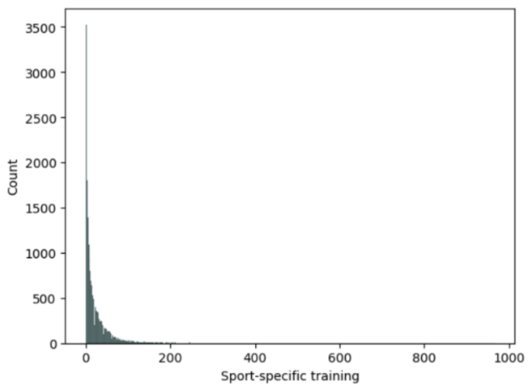
To understand the shape of variables distribution, the **skewness** was calculated for each of the numeric variables included in our dataset and plotted the respective histograms.

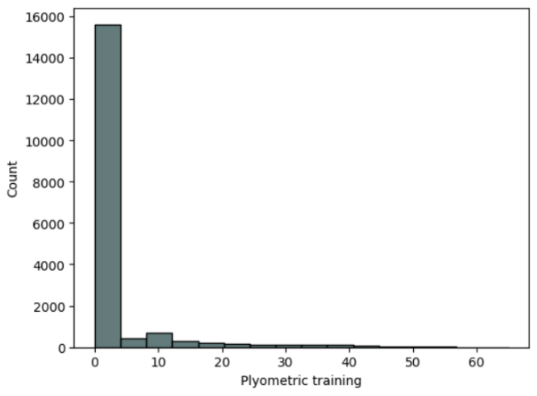


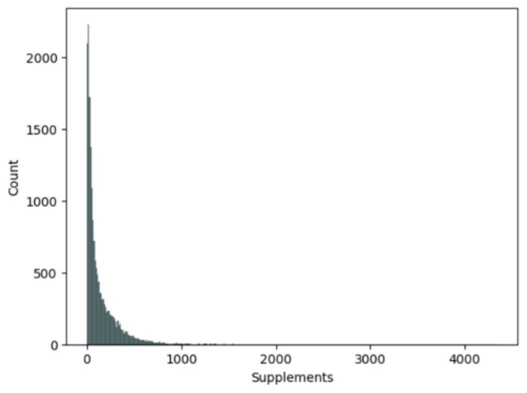
```
RecordID                    0.007411
Edition                    -0.230253
Athlete Id                  2.432872
Previous attempts           3.910222
Athlete score               1.182874
Train bf competition        4.543343
Strength training           3.124692
Sand training              35.498272
Recovery                    6.349221
Supplements                 3.519758
Cardiovascular training     5.559093
Squad training              7.064310
Physiotherapy               5.624916
Plyometric training         3.845004
Sport-specific training     7.124256
Other training              5.803005
Outcome                    -0.392536
dtype: float64
```

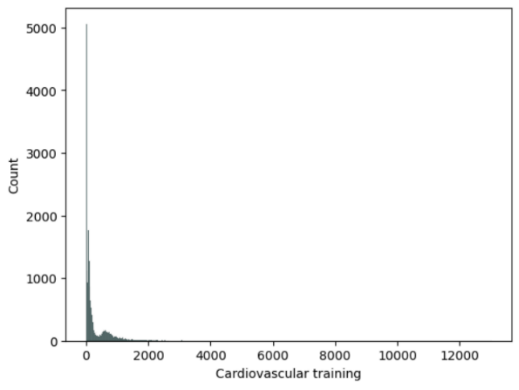
Table 6 Variable´s Skewness Value

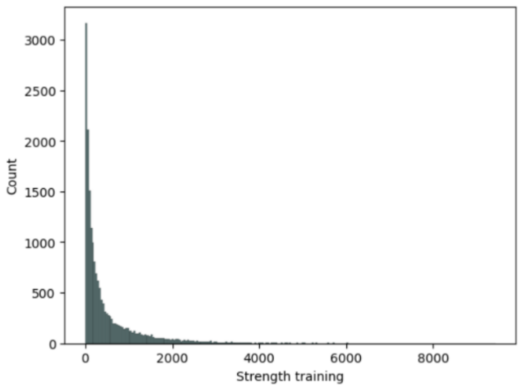
Based on the results obtained, it was verified that it would be unreasonable to assume a normal distribution of the data. However, since none of the models that will be used assume that the data follow a Gaussian distribution, we chose not to treat the skewness.

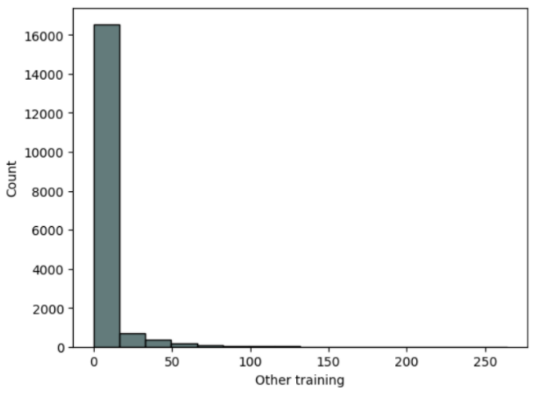


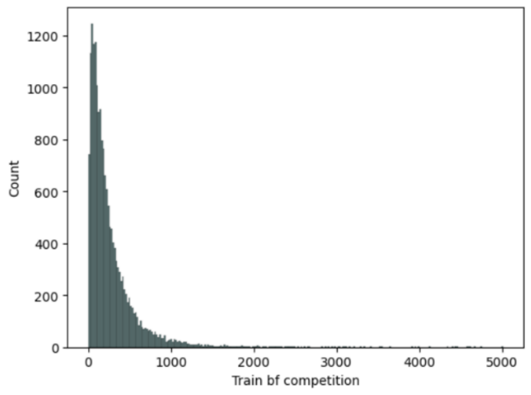


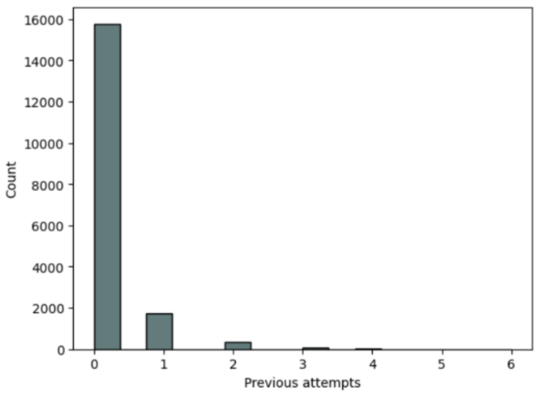


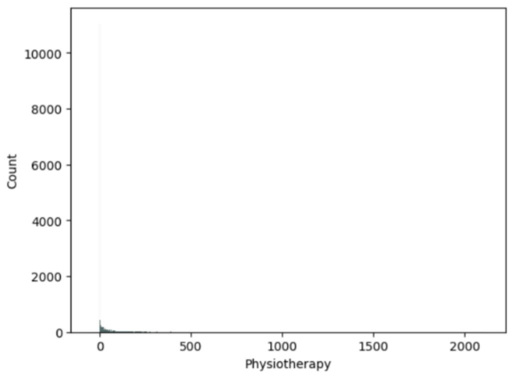


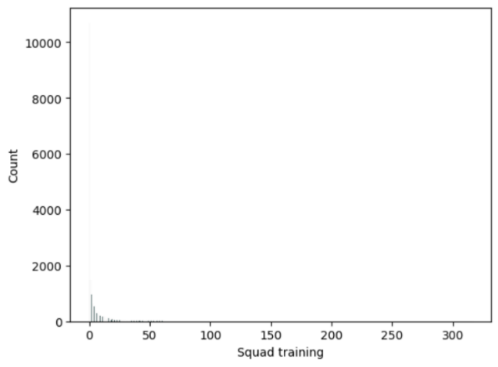

Count
Strength training

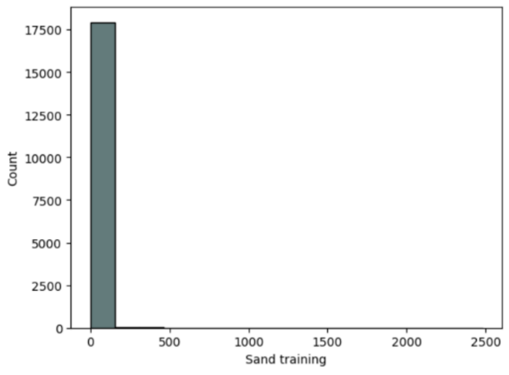

Count
Other training

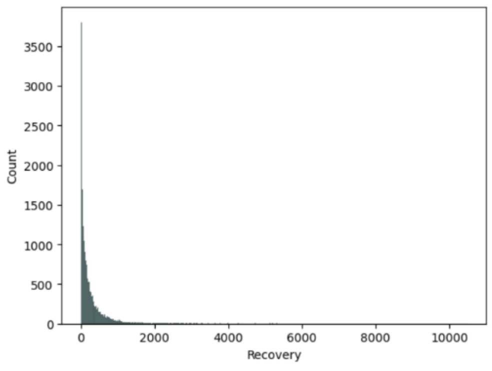

Count
Train bf competition


Count
Previous attempts


Count
Physiotherapy


Count
Squad training


Count
Sand training


Count
Recovery

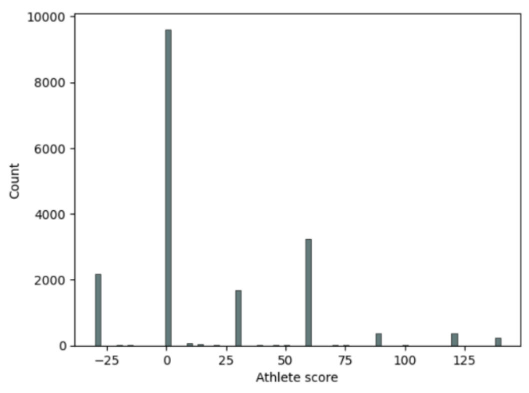


Figure 3 Histograms

4. Data Split

Before making modifications to the dataset, we should have three different datasets: **train, validation,** and **test** so that the evaluation of our models does not get biased. We already have a separated file with the test data, so we only split the train data into train and validation.

5. Pre-processing

Pre-processing is a crucial step in machine learning since it ensures good data quality prior to running algorithms. Thus, in this chapter we are going to explain how we solved our data issues in order to **improve our models' performance**, trying to reach more accurate predictions. The decisions made at this stage were exclusively informed by the training data.

5.1. Check for Duplicates

Firstly, we **checked for duplicates** in the three datasets and did not find any duplicated rows.

5.2. Set RecordID as Index

Since the RecordID is a special key that identifies each athlete into an edition of a given competition, rather than an attribute, we converted it to the **index**.

5.3. Incoherences

For the categorical variables, we checked if there were any errors or incoherent values in the column by analyzing the **value counts**. For the numerical ones, we just checked the **maximum and the minimum values**.

- The first incoherence we detected was in the variable ***Age group***. The label for people who are more than 55 years old should be "55>=" instead of "55<=". To treat this incoherence, we replaced the original label with the correct one in the three datasets. Additionally, in train and validation datasets there was a label exclusively for athletes with 0 years old. Since this does not make any sense, we joined this category to the one that has the younger athletes, which is 0-35.

- Secondly, when checking the maximum and minimum values for the ***Athlete score****,* we found out that there were negative values. In sport competitions there are not negative scores. This way, we replaced all the values for this variable in the three datasets by their absolute values.

- The following incoherence we detected was in ***Mental Preparation*** variable. This is a boolean variable so it only has two possible values: True or False. Thus, the values labeled as "FASE" should be "FALSE". We also replaced the labels "TRUE" and "FALSE" by "True" and "False", respectively, to be coherent with the rest of the other boolean variables of our dataset.

- Finally, the last incoherence was on the variable ***Physiotherapy*** and is similar to the one detected in the variable *Athlete score*. It is impossible to have negative values for the number of physiotherapy sessions.

Therefore, we use the absolute value function in the three datasets to transform all values of *Physiotherapy* in positive ones.

5.4. Outliers

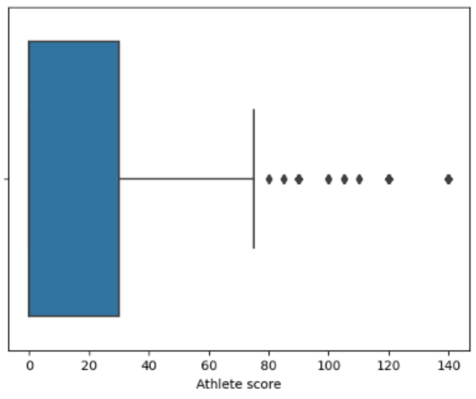
Outliers are observations that lie an abnormal distance from other values of the dataset. These values can compromise the database and the project objectives. At this stage, all decisions were exclusively informed by the training data.

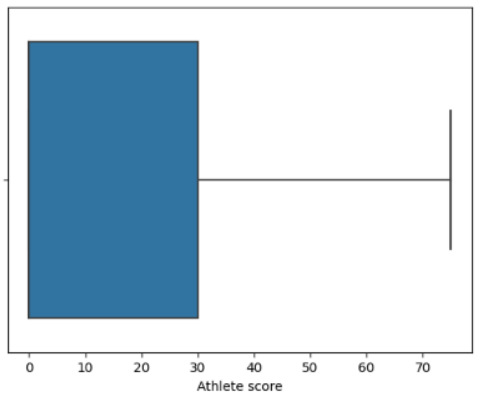
We defined two different approaches to treat outliers:

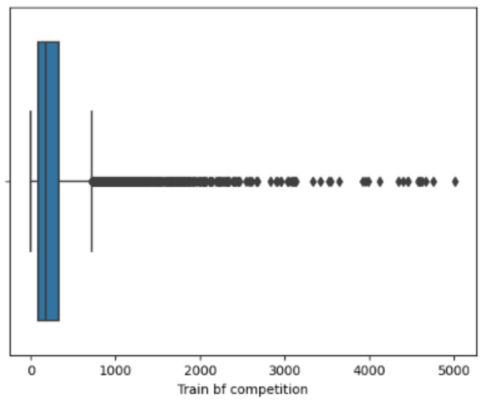
- The first one was used in the variables *Athlete score*, *Train bf competition*, *Strength training*, *Recovery*, S*upplements*, *Cardiovascular training*, *Squad* training, Physiotherapy and *Sport-specific training*. We realized through the **interquartile** rule, that there were too many values considered as outliers that could not just be eliminated, so we decided to treat and **replace** them for these variables. For this, we defined a maximum limit and a minimum limit for each of the variables with outliers. We established an upper limit, which is determined by subtracting 1.5 times the interquartile range (**IQR**) from the first quartile (**Q1**), and a lower limit, which is calculated by adding 1.5 times the IQR to **Q3**. Each value that was greater than the upper limit was replaced by the upper limit and each value that was less than the lower limit was replaced by the lower limit of the respective variable.

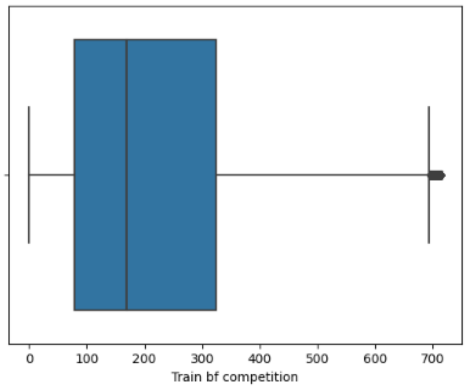
- For the variables *Sand training*, *Other training* and *Plyometric training* we defined a **threshold** by looking to the boxplot of each variable and eliminated the values that exceeded the threshold. It is important to note that we did not eliminate more that 3% of the dataset.

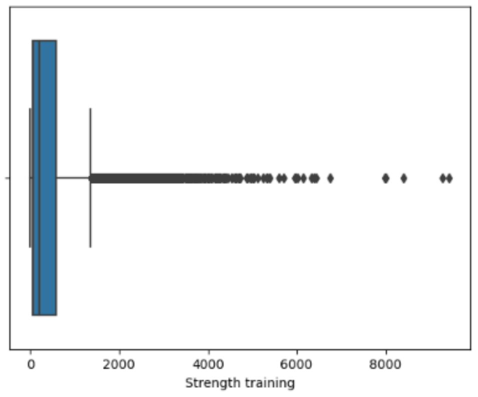
In order to visualize the outliers, we plotted boxplots for the numerical variables before and after treating outliers.

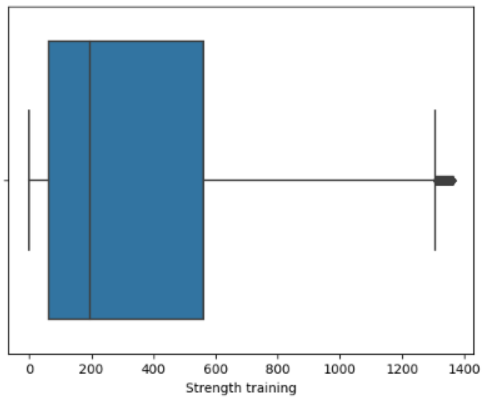

0
20
40
60
80
100
120
140
Athlete score

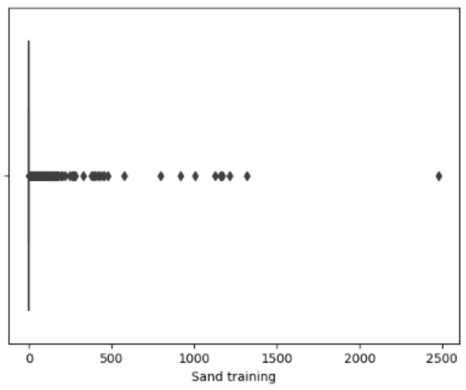

0
10
20
30
40
50
60
70
Athlete score

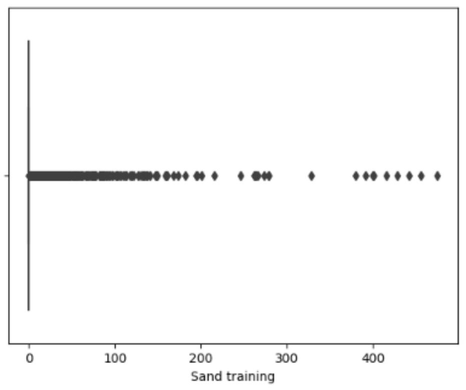

0
1000
2000
3000
4000
5000
Train bf competition

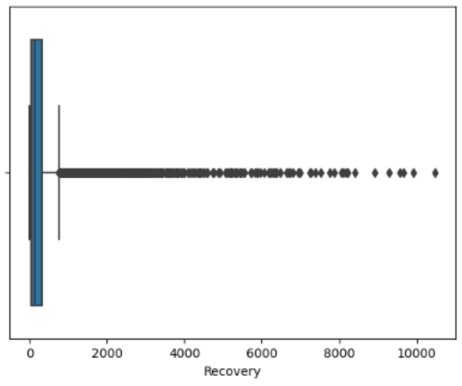

0
100
200
300
400
500
600
700
Train bf competition

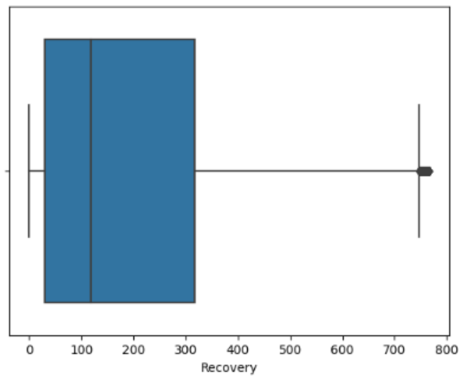

0
2000
4000
6000
8000
Strength training

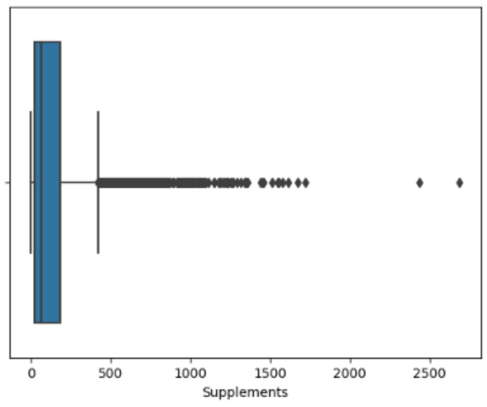

0
200
400
600
800
1000
1200
1400
Strength training

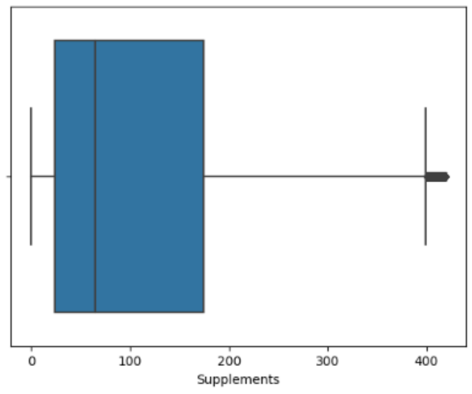

0
500
1000
1500
2000
2500
Sand training

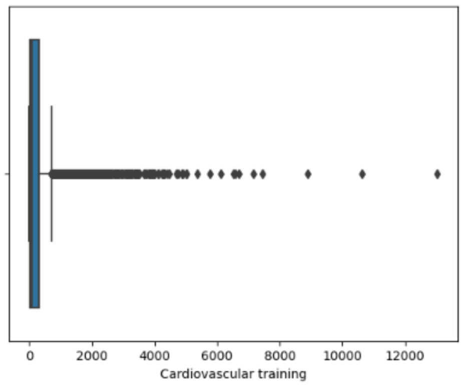

0
100
200
300
400
Sand training

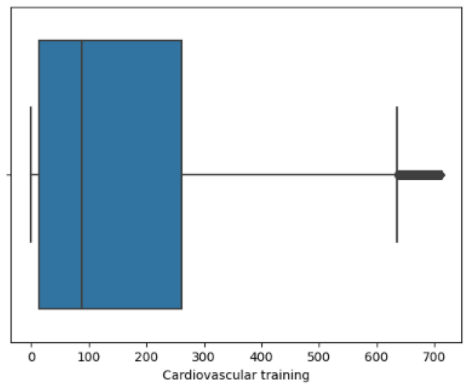

0
2000
4000
6000
8000
10000
Recovery

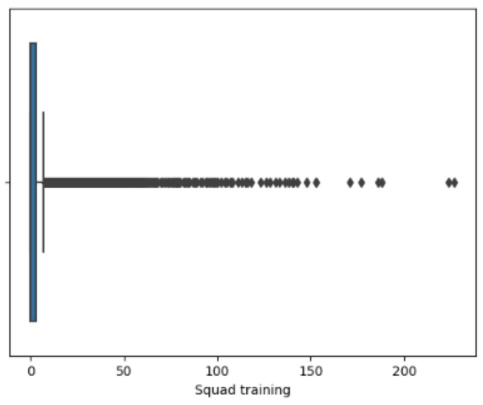

0
100
200
300
400
500
600
700
800
Recovery

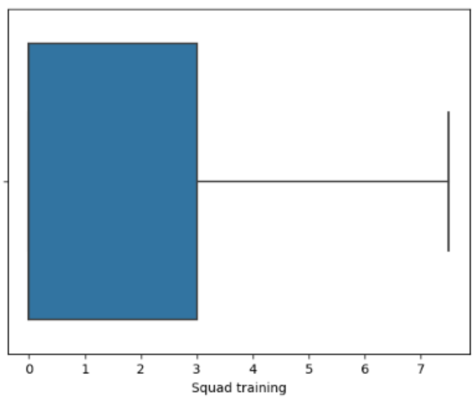

0
500
1000
1500
2000
2500
Supplements

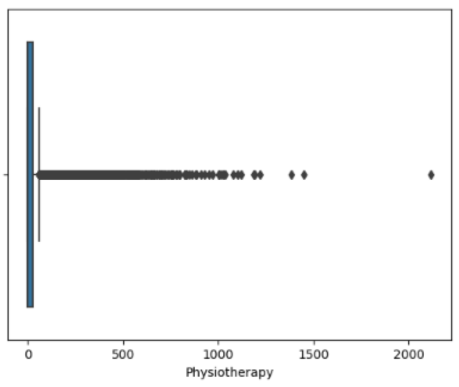

0
100
200
300
400
Supplements

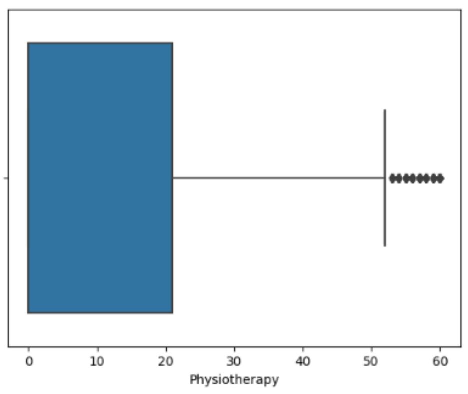

0
2000
4000
6000
8000
10000
12000
Cardiovascular training

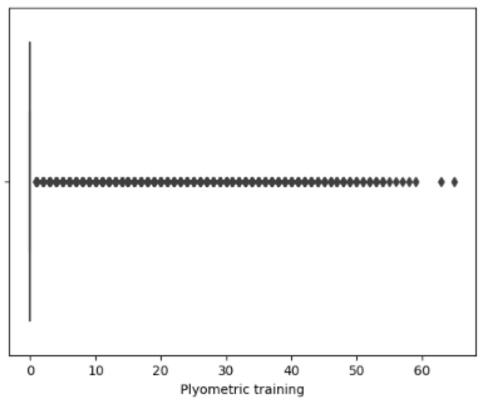

0
100
200
300
400
500
600
700
Cardiovascular training

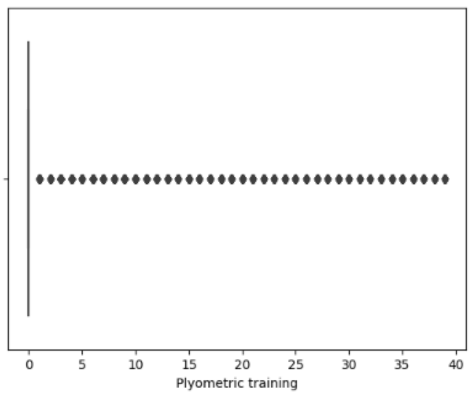

0
50
100
150
200
Squad training


0
1
2
3
4
5
6
7
Squad training


0
500
1000
1500
2000
Physiotherapy


0
10
20
30
40
50
60
Physiotherapy


0
10
20
30
40
50
60
Plyometric training


0
5
10
15
20
25
30
35
40
Plyometric training

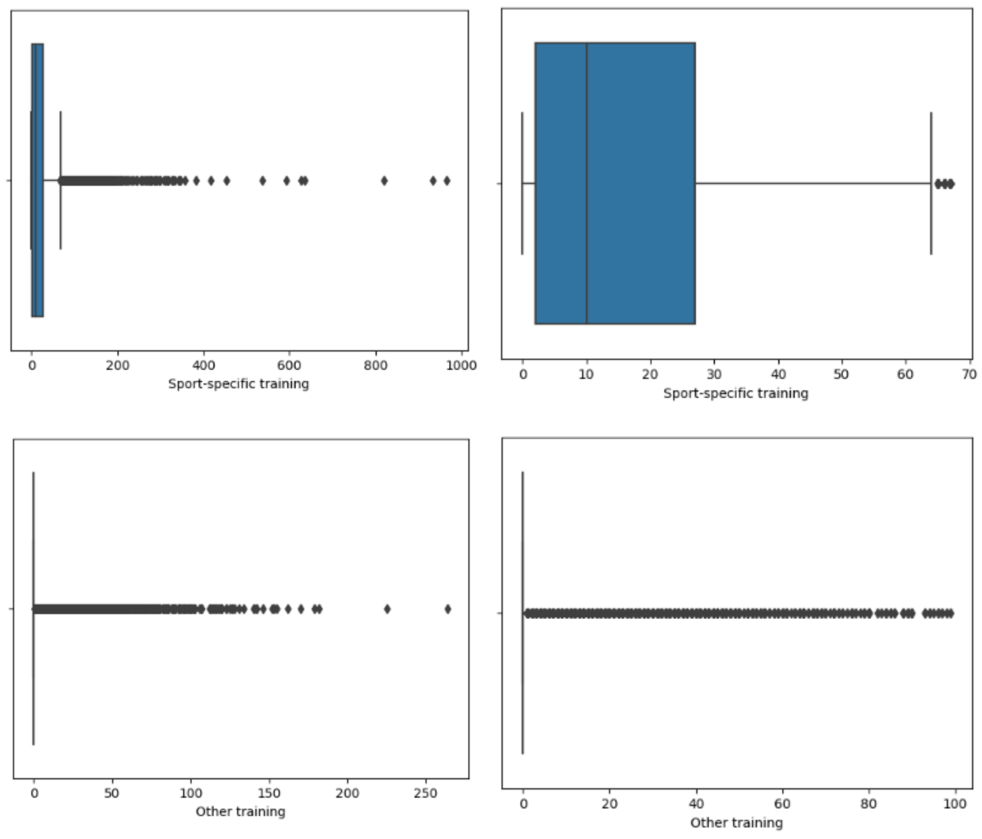
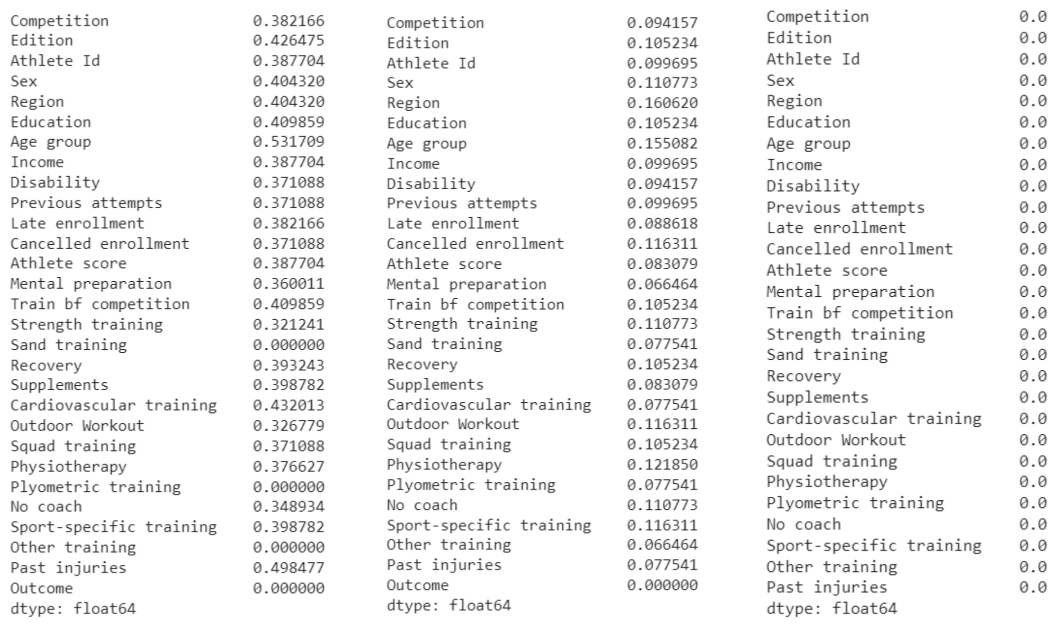


Figure 4 Boxplots for Outliers Identification (Before and After Treating)

5.5. Missing Values

Missing values, which refer to absent or unknown data points, require appropriate handling to ensure the integrity and reliability of data mining analyses.

We analyzed the **percentage of missing values** of the variables of each of the three datasets and concluded that the test dataset had no missing values. In contrast, the train and the validation datasets had variables with some missing values, but any variable exceeded 5% of missing values. The figures below show the percentage of missing values in each variable of the three datasets.



```
Competition                0.382166
Edition                    0.426475
Athlete Id                 0.387704
Sex                        0.404320
Region                     0.404320
Education                  0.409859
Age group                  0.531709
Income                     0.387704
Disability                 0.371088
Previous attempts          0.371088
Late enrollment            0.382166
Cancelled enrollment       0.371088
Athlete score              0.387704
Mental preparation         0.360011
Train bf competition       0.409859
Strength training          0.321241
Sand training              0.000000
Recovery                   0.393243
Supplements                0.398782
Cardiovascular training    0.432013
Outdoor Workout            0.326779
Squad training             0.371088
Physiotherapy              0.376627
Plyometric training        0.000000
No coach                   0.348934
Sport-specific training    0.398782
Other training             0.000000
Past injuries              0.498477
Outcome                    0.000000
dtype: float64
```

```
Competition                0.094157
Edition                    0.105234
Athlete Id                 0.099695
Sex                        0.110773
Region                     0.160620
Education                  0.105234
Age group                  0.155082
Income                     0.099695
Disability                 0.094157
Previous attempts          0.099695
Late enrollment            0.088618
Cancelled enrollment       0.116311
Athlete score              0.083079
Mental preparation         0.066464
Train bf competition       0.105234
Strength training          0.110773
Sand training              0.077541
Recovery                   0.105234
Supplements                0.083079
Cardiovascular training    0.077541
Outdoor Workout            0.116311
Squad training             0.105234
Physiotherapy              0.121850
Plyometric training        0.077541
No coach                   0.110773
Sport-specific training    0.116311
Other training             0.066464
Past injuries              0.077541
Outcome                    0.000000
dtype: float64
```

```
Competition                0.0
Edition                    0.0
Athlete Id                 0.0
Sex                        0.0
Region                     0.0
Education                  0.0
Age group                  0.0
Income                     0.0
Disability                 0.0
Previous attempts          0.0
Late enrollment            0.0
Cancelled enrollment       0.0
Athlete score              0.0
Mental preparation         0.0
Train bf competition       0.0
Strength training          0.0
Sand training              0.0
Recovery                   0.0
Supplements                0.0
Cardiovascular training    0.0
Outdoor Workout            0.0
Squad training             0.0
Physiotherapy              0.0
Plyometric training        0.0
No coach                   0.0
Sport-specific training    0.0
Other training             0.0
Past injuries              0.0
dtype: float64
```

Table 7 Missing Values for Train, Validation and Test

We developed two different approaches when dealing with missing values: one for the numerical variables and other for the categorical ones. For the numeric features we filled the missing values with the **median value** and for the categorical ones we used the **mode value**.

5.6. Feature Engineering

5.6.1. Data Transformations

Feature engineering is the process of manipulating and transforming raw data into features that can be used to **boost our machine learning models**. Thus, our group applied some feature engineering processes, that are described below. It is important to note that all these transformations were made to the three datasets: train, validation and test.

- For the variable *Education*, we considered the label *Post Graduate* very similar to the *University Degree* one. Hence, we replaced *Post Graduate* by *University Degree*, to reduce the number of categories.

- In the feature *Income*, as the labels were too detailed, we joined *Middle-Low* with *Middle-High* creating a new label called *Middle*.

- The variable *Region* had too many labels; hence, we joined some labels to reduce them. We replaced *North America*, *South America* and *Central America* by *America*. We also replaced *Western Europe*, *Eastern Europe* and *Southern Europe* by *Europe*. Then, we replaced *East Asia*, *South Asia* and *Central Asia* by *Asia*. Finally, we replaced the labels *Northern Africa* and *Southern Africa* by *Africa*.

- Lastly, there were some nearly unary variables, such as *Mental preparation*, *Outdoor workout*, *Late enrollment* and *No coach*. In these variables, one category has a significantly smaller proportion of instances compared to the other category. Thus, we dropped them as we considered them not to be relevant in our analysis.

5.6.2. Encoding Categorical Data

Another important step in feature engineering is the process of transforming categorical variables into a numerical form. This is crucial, as it allows predictive models to understand and learn from the data more effectively.

First, we applied **Ordinal Encoding** to the variable *Age group* with the purpose of assigning numerical values to each category based on its order and to maintain the hierarchical relationship between them. Then we fitted the encoder to the training set and transformed the training, validation and test sets.

Next, we decided to use **Target Encoding** for the variables that had many categories, such as *Competition, Region, Education* and *Performance.* Instead of creating a large number of binary dummy variables, this technique replaced each category with the mean of the target variable *Outcome* which helps reduce dimensionality of the data. Similar to the technique above, we fitted the encoder on the train set and apply it to all sets.

Lastly, we used **dummy variables** to recode the variables that have two categories, such as *Sex, Disability, Cancelled enrollment* and *Previous injuries* into flag variables that take only two values, 0 and 1. This process was applied to the training, validation and test sets.

5.7. Scaling Data

After exploring the data, we reached the conclusion that the variables have different ranges, which can affect the performance of the algorithms on the modelling stage.

To ensure that all variables are consistent we decided to employ the **MinMaxScaler** to scale the data. This scaler captures the range of each feature in the training set and will then be used to transform all datasets.

It is important to mention that the dependent and independent variables were split before the scaling and merged in new dataset after the scaling.

5.8. Feature Selection

Feature selection is a crucial task in data mining projects that involves the selection of a subset of the most relevant variables with the goal of creating effective models with high performance and to reduce the risk of overfitting.

In this section, we will present four feature selection techniques used in the project: **Spearman's Correlation**, **ANOVA** (Analysis of Variance), **Chi-square**, and **Recursive Feature Elimination** (RFE).

5.8.1. Spearman Correlation

Firstly, we have opted for **Spearman's correlation** over Pearson's correlation as it does not assume a normal distribution of the data, considers linear and non-linear relationships between variables and is also more robust to outliers, therefore being a better fit for our dataset.

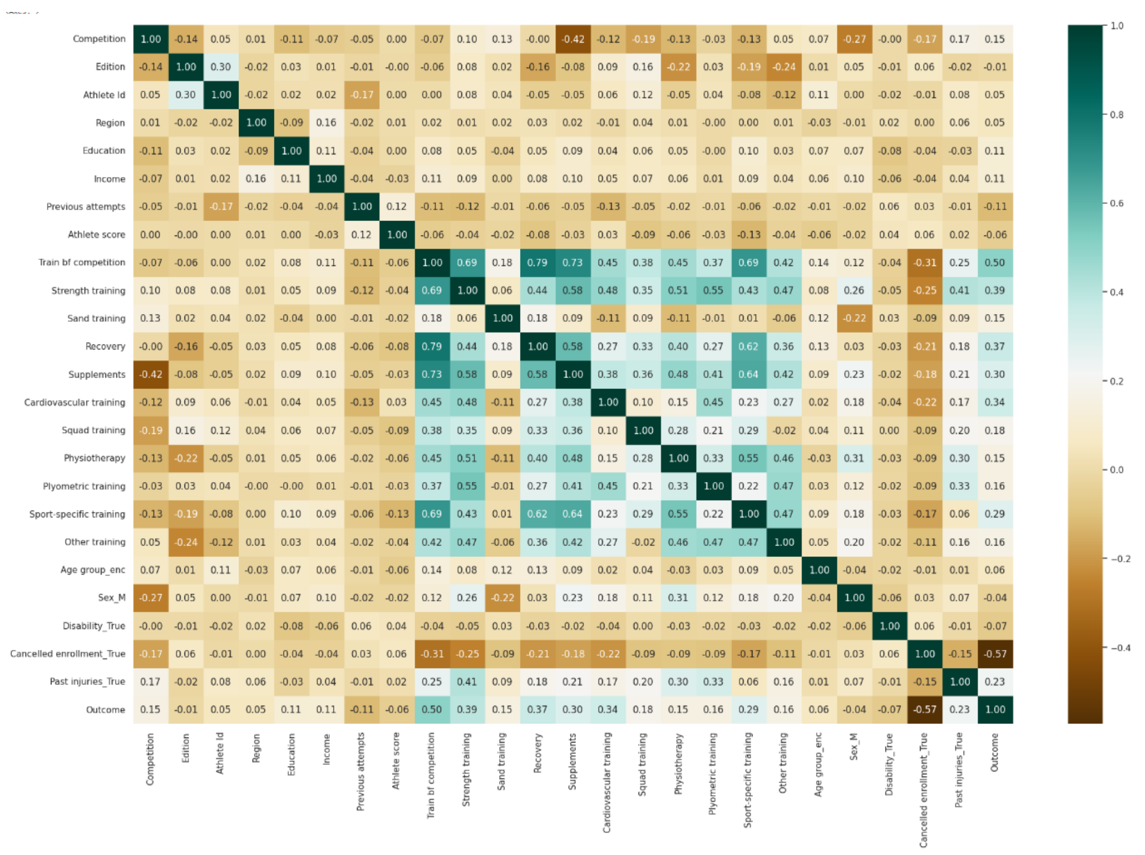


Figure 5 Spearman Correlation Matrix

To identify variables with strong correlation to the target variable, a **correlation matrix** was plotted, and threshold values to define strong relationships established: higher than 0,5 or lower than -0,5.

Based on this, only two variables were selected: *Cancelled enrollment_True* and *Train bf Competition*.

5.8.2. ANOVA

Next, the **ANOVA** method was applied to relate the numerical variables to the categorical output.

Thus, we developed a **loop** to find the best number of features that maximize the F1-score in the validation dataset. The number of features chosen was 10.

Subsequently, **ANOVA with k=10** was applied to determine which specific features should be included in the model. The selected features were *Previous attempts, Competition training, Strength training, Supplements, Cardiovascular* *training, Team training, Physiotherapy, Plyometric training, Sport specific training* and *Other training.*

By incorporating these selected features, the performance and predictive ability of our logistic regression model is enhanced.

5.8.3. Chi-Square

Then, the **Chi-Square** method was employed to check if the categorical variables have a **significant relationship with the target**. The Chi-Square test was then performed, and it stated that the null hypothesis assumes no relationship, while the alternative hypothesis suggests a relationship between the variables.

From the results of the Chi-Square test, it was found that the variables with significant relationship to the target, and therefore should be selected were: *Edition, Recovery, Competition, Sex_M, Region, Education, Income, Age group_enc, Disability_True, Cancelled enrollment_True* and *Past injuries_True*.

5.8.4. Recursive Feature Elimination

Finally, **Recursive Feature Elimination** (RFE) was applied using two algorithms as the base estimator: **Logistic Regression** and **Decision Tree Classifier**. Firstly, we applied these methods for all features and concluded that the best number of features provided by the loop were not the most adequate.

Afterwards, the variables selected by the **ANOVA** (numerical) and **Chi-Square** (categorical) were used in the loop, for both methods. The table below summarizes the results provided by each loop performed:

Method	**Number of Features**	**F1 Score**
Decision Tree Classifier (all features)	9	0.8365
Logistic Regression (all features)	24	0.8699
Logistic Regression (ANOVA & Chi-Square)	19	0.8693
Decision Tree Classifier (ANOVA & Chi-Square)	2	0.8590

Table 8 Selection of number of features in RFE

The chosen method was the Logistic Regression employed with the features previously selected by the ANOVA and Chi-Square methods. Even though the Logistic Regression using all features achieved the highest F1 score, the approach using ANOVA and Chi-square provides a more adequate number of features as well as a high **F1 score**.

Subsequently, we performed **RFE with k=19** to identify which features should be included in the model. The selected features were *Competition, Edition, Region, Education, Income, Previous attempts, Train bf competition, Sand training, Recovery, Supplements, Cardiovascular training, Squad training, Physiotherapy, Plyometric training, Sport-specific training, Other training, Sex_M, Cancelled enrollment_True and Past injuries_True.*

6. Modeling

This chapter focuses on **exploration** and **evaluation** of the various machine learning **algorithms** applied to our dataset. Each one of the models presented below was **trained on the provided training data**, allowing it to learn the underlying patterns and relationships between the chosen features and the target variable *Outcome*.

6.1 Bayesian and Instance Based Leaning

Bayesian classifiers rely on **Bayes' theorem** to make predictions and are particularly useful when dealing with noisy data and incomplete information. This algorithm estimates the probability distribution of each class based on the features of the instances. When a new instance arrives, it calculates the probability of that instance belonging to each of the classes.

That said, this was the first algorithm to be applied to training and validation datasets, where we used the implementation associated with it, GaussianNB. To build the **modelNB** we decided to use the default parameters already embedded in it. However, the results obtained were not particularly interesting, having obtained a score of **0.7799 for the training dataset**, and a score of **0.7790 for the validation dataset**.

6.2 KNN Classifier

The second algorithm applied was **K-Nearest Neighbors**, which is known as a simple supervised learning algorithm and operates based on the principle of finding the "k" closest training instances and then making predictions based on the majority vote. It is also worth mentioning that KNN is a non-parametric algorithm since it does not make any assumptions about the underlying data distribution.

To begin with, a model called **modelKNN** was created with the default parameters defined by KNeighborsClassifier. After obtaining the **scores of 0.8738 and 0.8277 for the train and validation datasets,** respectively, it was found interesting to make some adjustments to the parameters so that the model could improve its results. For this reason, we resorted to a **loop** to identify which number of “k” would give better results.

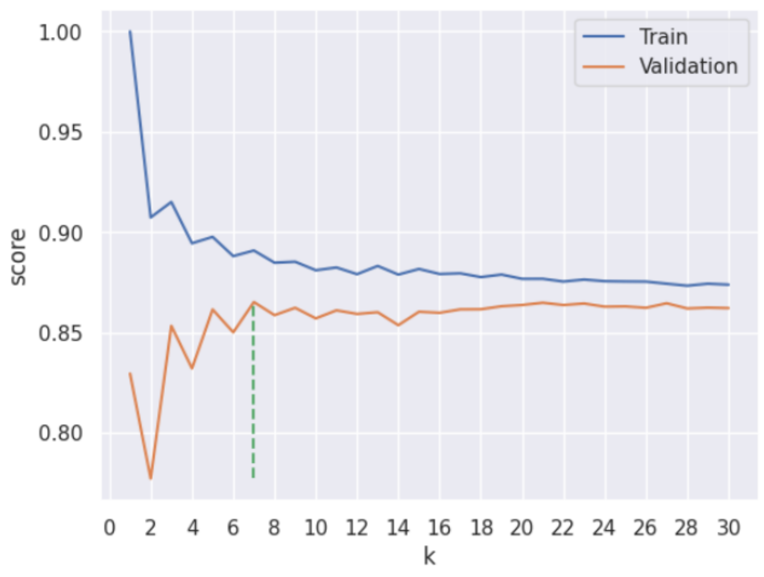


Figure 6 Finding the Best Number of K

From the graph above, the best model would have a value of "k" equal to 7. This way, we created **modelKNN7**, with k=7. Although the **score for the train dataset** obtained with this number of “k” has reduced (**0.8647**), the **score value obtained for the validation dataset has improved** (**0.8311**), even if they were almost insignificant variations.

Later, two more models were used with some variants in the parameters used. First, the distance metric used was changed to *manhattan*, instead of the *euclidean*, used by default. With this variation, we obtained the best score for the **modelKNNM**, with a **score of 0.8951 for the train and 0.8657 for the validation datasets**. Finally, we tried yet another variation, where this time all the neighbors would have the same weight, when using the parameter *weights=’distance’*. This last instance of the model, **modelKNNW,** turned out to be overfitting (with **a score of 1 for the train** dataset and a **score of 0.8658 for the validation one**), possibly because giving more weight to closer neighbors can result in a model that is too sensitive to local fluctuations and eventual noise in the data.

6.3 Decision Trees

The next algorithm applied was the **Decision Tree algorithm**. Decision trees are classification tools that discriminate between classes effectively by representing simple, interpretable rules.

The initial Decision Tree model, **modelDT***,* was trained with the default parameters, achieving a **score of 1.0 on the training dataset**, indicating potential **overfitting**. On the **validation dataset**, however, the model's score was **0.8022**.

Then, a second decision tree, **modelDT_entropy**, was trained using another splitting criterion, the **entropy for information gain**. While the scores remained the same for the **training data set (1.0)**, the **validation score** decreased slightly to **0.8017**.

Therefore, both decision trees displayed **signs of overfitting**, suggesting that more tuning was required.

To address the overfitting problem and improve our results, we used the **prepruning** approach, which stops growing the tree earlier, before it perfectly classifies the training set. So, we adjusted the parameters by varying the **minimum samples needed to split** and the **maximum depth** of the decision tree.

A loop was implemented to assess several combinations and it was determined that the parameters that provided the **highest validation score of 0.8629** had **min_samples_split=10** and **max_depth=7.**

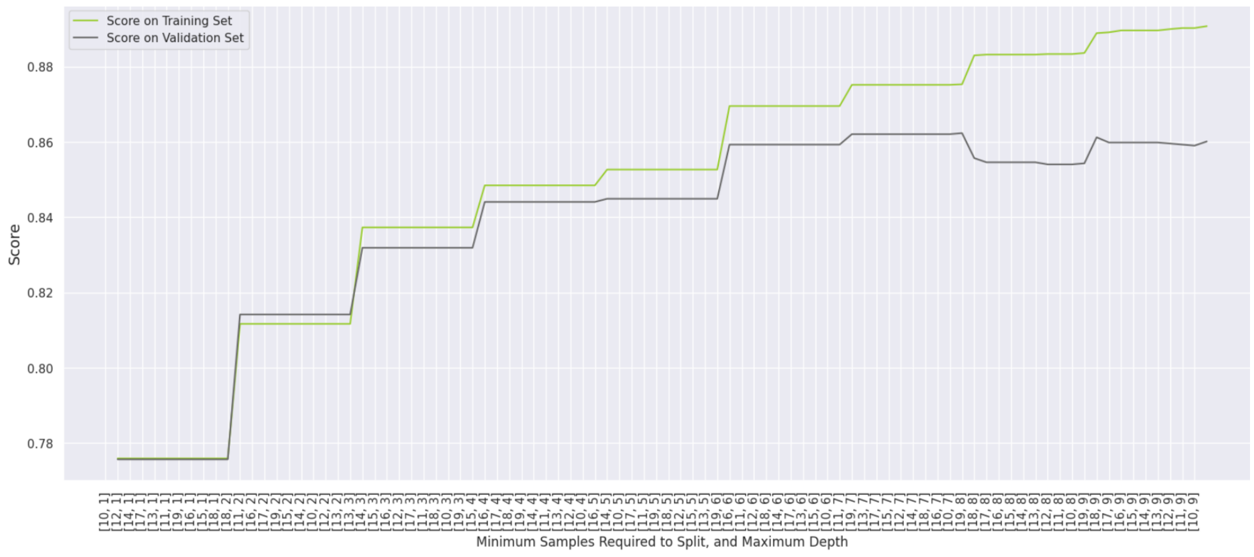


Figure 7 Finding the minimum samples required to split and maximum depth

Ultimately, a new decision tree model, **modelDT_chosen_parameters,** was developed using the selected parameters, which provided a **training score of 0.8753** and a **validation score of 0.8621**. These results show that the model has the ability to generalize well to unseen data.

6.4 Ensemble Classifiers

Random Forest

The **random forest** algorithm is an **ensemble classifier** that combines multiple decision trees to make predictions, thereby resulting in improved accuracy.

The initial model, **modelRF*,*** with the default parameters got a **training score of 1.0** and a **validation score of 0.8759**, which means that this model was **overfitting**.

To improve the results, a **grid search** with 10-fold cross-validation was performed to identify the best hyperparameters for the **Random Forest** model - **modelRF_parameters10**. First, it was performed with only two parameters, obtaining the following **n_estimators=150** and **max_depth=15.** This provided a

train score of 0.9705 and a **validation score of 0.8751,** which still showed signs of overfitting.

To further improve the performance, we did a grid search with three parameters: **n_estimators**, **max_depth** and **min_samples_split**. The best parameters obtained were **n_estimators=150**, **max_depth=15** and **min_samples_split=10,** with a **validation score of 0.8761**.

Thus, the model - **modelRF_parameters_best** - was trained again with these optimal parameters achieving a **training score of 0.9295 and validation score of 0.8726**. Although the overfitting has been reduced, there remains some performance discrepancy between the training and validation scores. At this point we stopped optimizing the model since by adding more parameters, it took an excessive amount of time for the model to form an output.

Bagging

Bagging stands for Bootstrap Aggregation. It is an assembled method that manipulates and samples the training dataset into multiple subsets. The model is then trained in the different subsets independently and the results are averaged in case of regression and voted in case of classification.

To perform the bagging technique, we decided to use the Decision Tree as the base learner, model **bagging_DT**, in this case, we will use the Decision Tree that we created before. After training the model the score for the **training dataset was 0.9935** and for **validation was 0.8565**, which means that the model was **overfitting** since the score for the train dataset is very close to 1 and considerably far from the validation.

We decided to change the parameters looking for a better score. Since we created a decision tree with parameters chosen based on the plot analysis, we used it as a base estimator - model **bagging_DT_parameters**. After fitting and training the model the score for the **train dataset was 0.8786** and for **validation was 0.8626.**

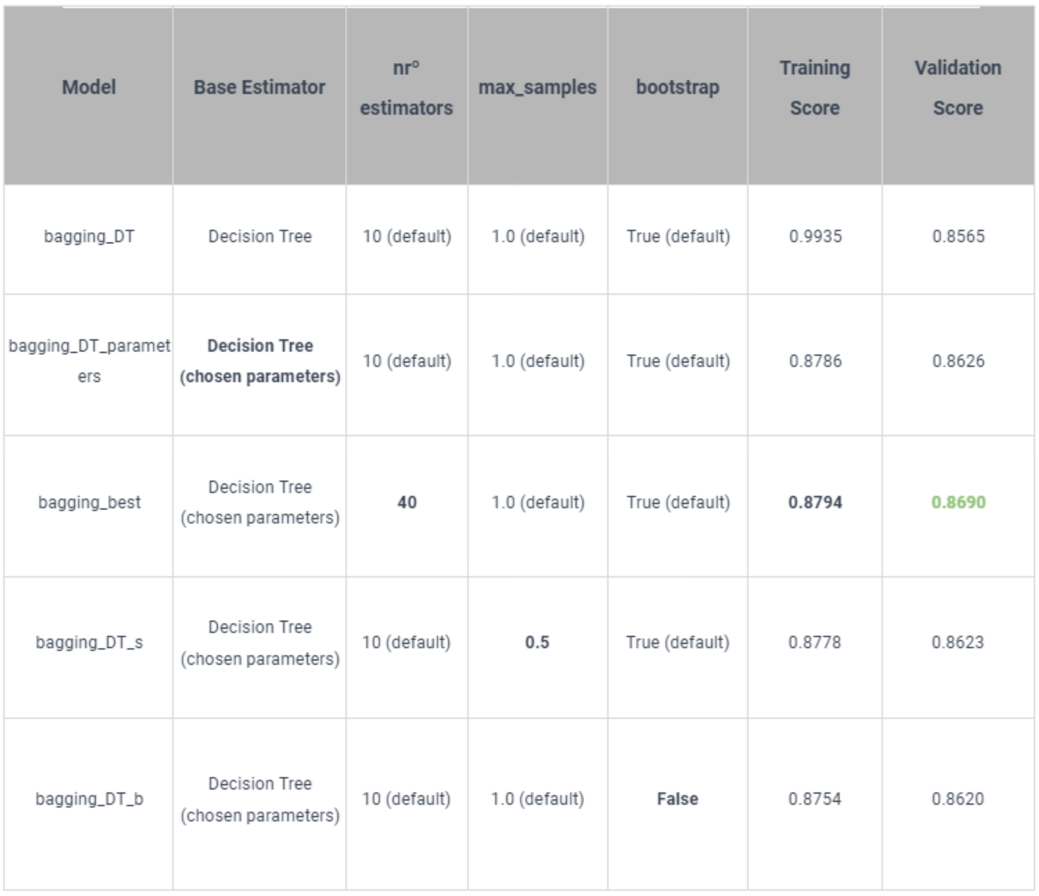
To improve the models' performance, we decided to perform a loop that would provide the best score model **bagging_best** that was not overfitting (threshold = 0.03).

Using a combination of the first Decision Tree we used and the one with the best parameters, with a list of estimators ranging from 20, 40, 50, 60 or 70. The score for the **training dataset was 0.8794** and for the **validation was 0.8690**.

Despite the better scores, we wanted to make use of the bootstrap possibilities. In the **BaggingClassifier** function, **max_samples** is by default 1, we decided to equal it to 0.5 meaning that each decision tree (our estimator) will only see half of the data randomly picked. We kept the **Decision Tree** with the modified parameters as a base estimator (model **bagging_DT_s**) and the score for the **train dataset was 0.8778** and for **validation 0.8623**

We decided to change the way that the samples were drawn, to do this we changed the parameter of Bootstrap to False (model **bagging_DT_b**). By defining it as False we were saying that we do not want our samples to be picked with replacement. The score for the **train dataset was 0.8754** and for **validation was 0.8620.**

Here is a table summarizing the explanation above:



Model	Base Estimator	nr° estimators	max_samples	bootstrap	Training Score	Validation Score
bagging_DT	Decision Tree	10 (default)	1.0 (default)	True (default)	0.9935	0.8565
bagging_DT_parameters	**Decision Tree (chosen parameters)**	10 (default)	1.0 (default)	True (default)	0.8786	0.8626
bagging_best	Decision Tree (chosen parameters)	**40**	1.0 (default)	True (default)	**0.8794**	**0.8690**
bagging_DT_s	Decision Tree (chosen parameters)	10 (default)	**0.5**	True (default)	0.8778	0.8623
bagging_DT_b	Decision Tree (chosen parameters)	10 (default)	1.0 (default)	**False**	0.8754	0.8620

Table 9 Bagging models' overview

The group decided to explore a little further trying to improve the model performance. We made a loop and created a new model, **bagging_news,** where we provide a list of base estimators, being these **KNeighborsClassifier, RandomForestClassifier, LogisticRegression, GaussianNB** and **Support Vector Classifie**r (SVC) and a list with the different options of **numbers of estimators**, being 20, 40, 50, 60 and 70.

This loop gave us the **best validation score** possible between the combination of the list of base estimators and the available number of estimators that would not overfit (threshold = 0.03).

The result showed that the model **bagging_news**, with the best **validation score was 0.8332**, with **SVC as a base estimator** with a **train score of 0.8563.** Additionally, there is a theoretical explanation provided in the annex about this extra classifier.

After all this analysis we can say that the model with better performance was the **B-Best,** with a **train score of 0.8794** and a **validation score of 0.8690.**

<u>XG-Bost – Extra Algortihm</u>

For the extra algorithm, the group chose **XGBost** and its specific implementation, **XGBClassifier**. In short, we chose XGBoost since it provides efficient and accurate ensemble-based classification, allowing to leverage its advanced techniques for improved predictive modeling. However, there is a theoretical explanation provided in the annex about this.

We started by creating a model with the **default parameters** of XGBClassifier, **model_xgb**. After training this model, we scored **0.9512 in the training data** and **0.8706 in the validation data**. These values indicated the **overfitting** of **model_xgb**. Therefore, we performed a **grid search** to find the **optimal parameter** values. The combination of parameters that performed the **best validation score** were 3 for **max_depth**; 1 for **min_child_weight**; 0.5 for **gamma**; 0.8 for **subsample** and 0.8 for **colsample_bytree**. Thus, we created a model with the best parameters and named it **model_xgb_best**. In this model, the **training score was 0.8911** and the **validation score was 0.8820**. This way, we were able to optimize our model and combat overfitting.

6.5 Neural Networks

Neural Networks have some of the benefits over the others related to the performance of the model. This one has the ability to **self-learn** from the inputs received, do **non-linear interpolation** and has a major capacity to recreate the logic behind the human brain.

Our analysis will always be related to some requirements or parts that compose the model, for **MLP** (Multi-Layer Perceptron) solution. From the creation of an **MLPClassifier** instance to the execution of the **Grid Search**, which will have as output the "Mean Accuracy", we tested some **parameters** that will increase or decrease the accuracy. We started by doing a **model_nn** with the default parameters that **scored 0.785 for train** and **0.86 for validation**, and after we went with **model_nn2** (with two hidden layers and one neuron) with a **score of 0.875 for train** and **0.86 for validation**. Our third approach played with combinations in the grid search until we get the best parameter, **model_clf**: *tanh* for the activation function, **hidden layer** sizes of 10 and 15 (2 only), **learning rate** of 0.01, and lastly *adam* solver. The **scores obtained for train** was **0.8813** and **for validation was 0.8516**.

7. Performance Assessment

The **performance** of each model will be evaluated using various metrics such as **accuracy**, **precision**, **recall**, and **F1-Score**. The model that achieves the highest accuracy score on the validation set and displays the most balanced performance between the training and validation sets will be chosen as the best one. In the following table, there is the correspondence between the name of the model and its correspondent abbreviation:

Model Name	**Abbreviation**
modelNB	NB
modelKNN	KNN
modelKNN7	KNN7
modelKNNM	KNNM
modelDT	DT
modelDT_entropy	DT-E
modelDT_chosen_parameters	DT-CP
modelRF	RF
modelRF_parameters10	RF-P10
modelRF_parameters_best	RF-PB
bagging_DT	B
bagging_DT_parameters	B-CP
bagging_best	B-Best
bagging_DT_s	B-S
bagging_DT_b	B-B
bagging_news	B-N
model_xgb	XGB

model_xgb_best	XGB-B
model_nn	NN
model_nn2	NN2
model_clf	NN-CLF

Table 11 Model's name and abbreviation

Accuracy

The metric accuracy assesses the proportion of events correctly identified (positive or negative) on all possible events.

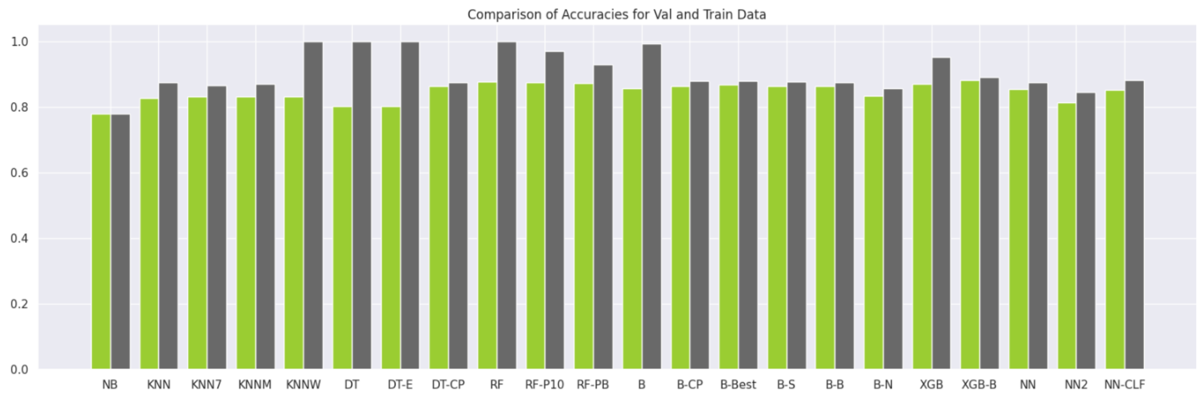
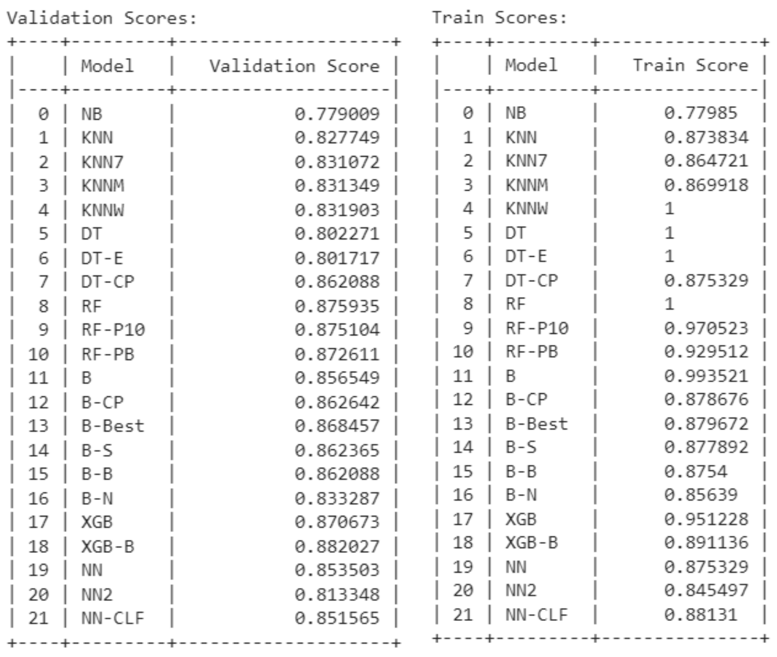


Figure 8 Comparison of Accuracy scores (green – val and grey - train)



```
Validation Scores:
+----+---------+--------------------+
|    | Model   |   Validation Score |
|----+---------+--------------------|
|  0 | NB      |           0.779009 |
|  1 | KNN     |           0.827749 |
|  2 | KNN7    |           0.831072 |
|  3 | KNNM    |           0.831349 |
|  4 | KNNW    |           0.831903 |
|  5 | DT      |           0.802271 |
|  6 | DT-E    |           0.801717 |
|  7 | DT-CP   |           0.862088 |
|  8 | RF      |           0.875935 |
|  9 | RF-P10  |           0.875104 |
| 10 | RF-PB   |           0.872611 |
| 11 | B       |           0.856549 |
| 12 | B-CP    |           0.862642 |
| 13 | B-Best  |           0.868457 |
| 14 | B-S     |           0.862365 |
| 15 | B-B     |           0.862088 |
| 16 | B-N     |           0.833287 |
| 17 | XGB     |           0.870673 |
| 18 | XGB-B   |           0.882027 |
| 19 | NN      |           0.853503 |
| 20 | NN2     |           0.813348 |
| 21 | NN-CLF  |           0.851565 |
+----+---------+--------------------+
```

```
Train Scores:
+----+---------+---------------+
|    | Model   |   Train Score |
|----+---------+---------------|
|  0 | NB      |      0.77985  |
|  1 | KNN     |      0.873834 |
|  2 | KNN7    |      0.864721 |
|  3 | KNNM    |      0.869918 |
|  4 | KNNW    |      1        |
|  5 | DT      |      1        |
|  6 | DT-E    |      1        |
|  7 | DT-CP   |      0.875329 |
|  8 | RF      |      1        |
|  9 | RF-P10  |      0.970523 |
| 10 | RF-PB   |      0.929512 |
| 11 | B       |      0.993521 |
| 12 | B-CP    |      0.878676 |
| 13 | B-Best  |      0.879672 |
| 14 | B-S     |      0.877892 |
| 15 | B-B     |      0.8754   |
| 16 | B-N     |      0.85639  |
| 17 | XGB     |      0.951228 |
| 18 | XGB-B   |      0.891136 |
| 19 | NN      |      0.875329 |
| 20 | NN2     |      0.845497 |
| 21 | NN-CLF  |      0.88131  |
+----+---------+---------------+
```

Table 10 Accuracy scores for validation and train datasets

Based on the graph and tables above, the models that achieved the **highest accuracy scores** on the validation dataset are:

- XGB-B: 0.882027
- RF: 0.875935
- RF-P10: 0.875104

Taking into account both validation scores and potential overfitting, **the best model seems to be XGB-B.** Since the train and validation score for this model are balanced, we can assume that it can generalize well to new, unseen data.

Precision

Precision measures the correctly predicted positive instances out of all positive predictions made by the model, which demonstrates how well the model predicts positive samples.

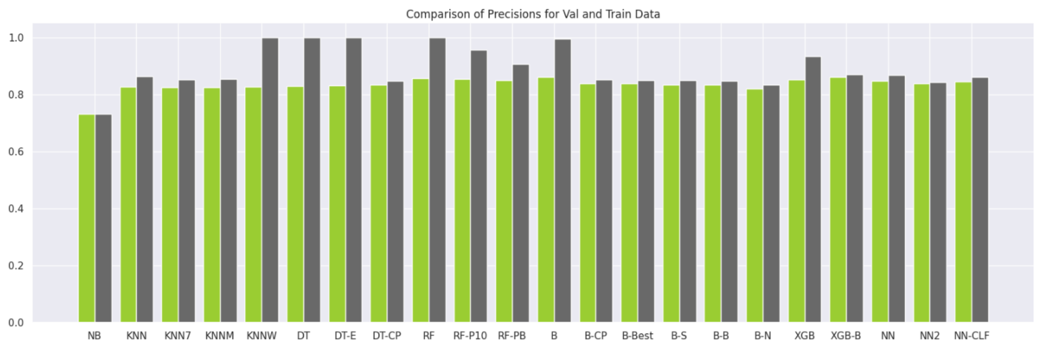
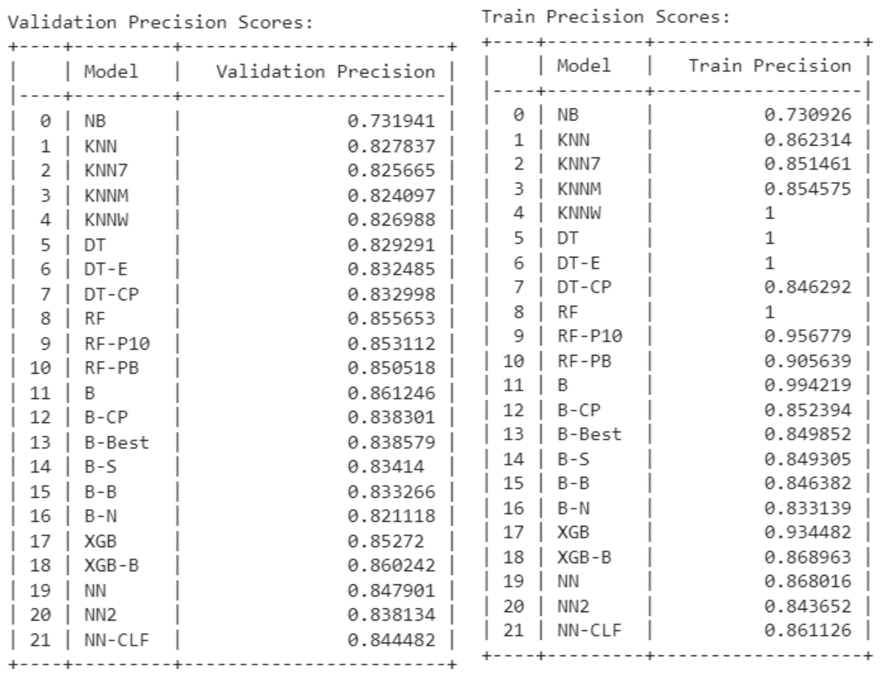


Figure 9 Comparison of Precision scores (green – val and grey - train)



```
Validation Precision Scores:
+----+---------+------------------------+
|    | Model   |   Validation Precision |
|----+---------+------------------------|
|  0 | NB      |               0.731941 |
|  1 | KNN     |               0.827837 |
|  2 | KNN7    |               0.825665 |
|  3 | KNNM    |               0.824097 |
|  4 | KNNW    |               0.826988 |
|  5 | DT      |               0.829291 |
|  6 | DT-E    |               0.832485 |
|  7 | DT-CP   |               0.832998 |
|  8 | RF      |               0.855653 |
|  9 | RF-P10  |               0.853112 |
| 10 | RF-PB   |               0.850518 |
| 11 | B       |               0.861246 |
| 12 | B-CP    |               0.838301 |
| 13 | B-Best  |               0.838579 |
| 14 | B-S     |               0.83414  |
| 15 | B-B     |               0.833266 |
| 16 | B-N     |               0.821118 |
| 17 | XGB     |               0.85272  |
| 18 | XGB-B   |               0.860242 |
| 19 | NN      |               0.847901 |
| 20 | NN2     |               0.838134 |
| 21 | NN-CLF  |               0.844482 |
+----+---------+------------------------+
```

```
Train Precision Scores:
+----+---------+-------------------+
|    | Model   |   Train Precision |
|----+---------+-------------------|
|  0 | NB      |          0.730926 |
|  1 | KNN     |          0.862314 |
|  2 | KNN7    |          0.851461 |
|  3 | KNNM    |          0.854575 |
|  4 | KNNW    |          1        |
|  5 | DT      |          1        |
|  6 | DT-E    |          1        |
|  7 | DT-CP   |          0.846292 |
|  8 | RF      |          1        |
|  9 | RF-P10  |          0.956779 |
| 10 | RF-PB   |          0.905639 |
| 11 | B       |          0.994219 |
| 12 | B-CP    |          0.852394 |
| 13 | B-Best  |          0.849852 |
| 14 | B-S     |          0.849305 |
| 15 | B-B     |          0.846382 |
| 16 | B-N     |          0.833139 |
| 17 | XGB     |          0.934482 |
| 18 | XGB-B   |          0.868963 |
| 19 | NN      |          0.868016 |
| 20 | NN2     |          0.843652 |
| 21 | NN-CLF  |          0.861126 |
+----+---------+-------------------+
```

Table 11 Precision scores for validation and train datasets

The models that achieved the **highest precision scores** on the validation dataset are:

- B: 0.861246
- XGB-B: 0.860242
- RF: 0.855653

Considering the validation precision scores and their potential for overfitting (perfect train scores), the **model XGB-B achieved the highest performance**, evidencing its ability to make accurate predictions.

<u>Recall</u>

Recall is a metric that measures the ability of a model to correctly detect positive instances and is used to measure the ratio of truly positive cases that are correctly classified.

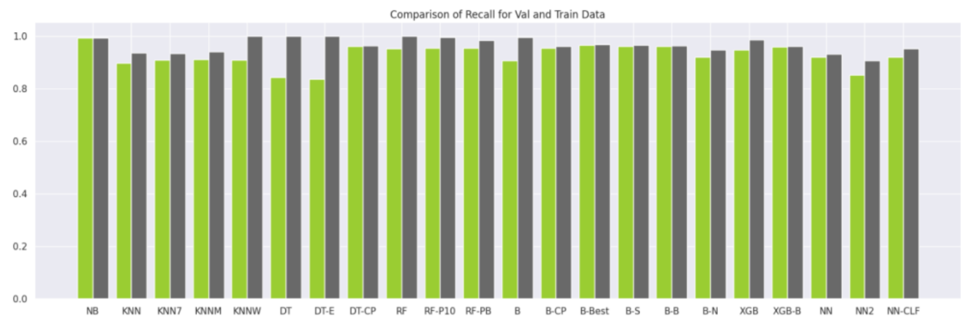
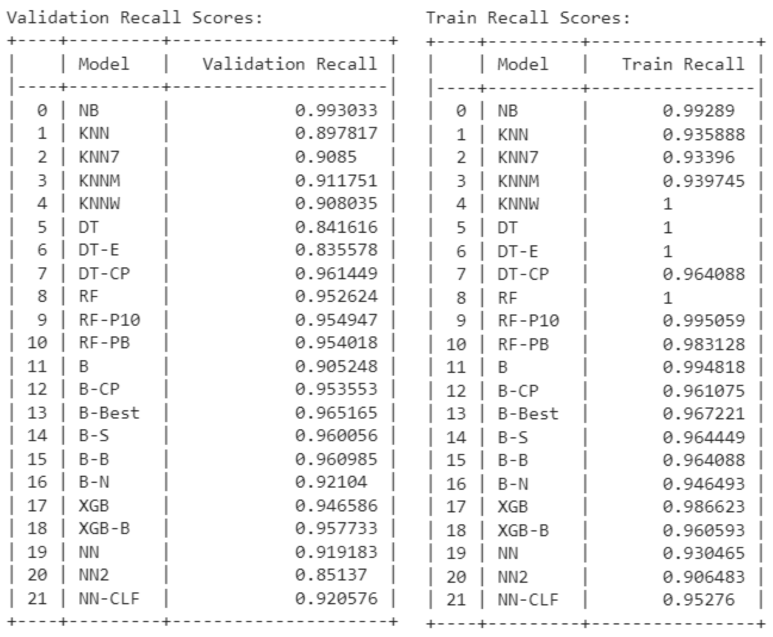


Figure 10 Comparison of Recall scores (green – val and grey - train)



```
Validation Recall Scores:                        Train Recall Scores:
+----+---------+--------------------+            +----+---------+----------------+
|    | Model   |   Validation Recall |           |    | Model   |   Train Recall |
|----+---------+--------------------|            |----+---------+----------------|
|  0 | NB      |           0.993033 |            |  0 | NB      |       0.99289  |
|  1 | KNN     |           0.897817 |            |  1 | KNN     |       0.935888 |
|  2 | KNN7    |           0.9085   |            |  2 | KNN7    |       0.93396  |
|  3 | KNNM    |           0.911751 |            |  3 | KNNM    |       0.939745 |
|  4 | KNNW    |           0.908035 |            |  4 | KNNW    |       1        |
|  5 | DT      |           0.841616 |            |  5 | DT      |       1        |
|  6 | DT-E    |           0.835578 |            |  6 | DT-E    |       1        |
|  7 | DT-CP   |           0.961449 |            |  7 | DT-CP   |       0.964088 |
|  8 | RF      |           0.952624 |            |  8 | RF      |       1        |
|  9 | RF-P10  |           0.954947 |            |  9 | RF-P10  |       0.995059 |
| 10 | RF-PB   |           0.954018 |            | 10 | RF-PB   |       0.983128 |
| 11 | B       |           0.905248 |            | 11 | B       |       0.994818 |
| 12 | B-CP    |           0.953553 |            | 12 | B-CP    |       0.961075 |
| 13 | B-Best  |           0.965165 |            | 13 | B-Best  |       0.967221 |
| 14 | B-S     |           0.960056 |            | 14 | B-S     |       0.964449 |
| 15 | B-B     |           0.960985 |            | 15 | B-B     |       0.964088 |
| 16 | B-N     |           0.92104  |            | 16 | B-N     |       0.946493 |
| 17 | XGB     |           0.946586 |            | 17 | XGB     |       0.986623 |
| 18 | XGB-B   |           0.957733 |            | 18 | XGB-B   |       0.960593 |
| 19 | NN      |           0.919183 |            | 19 | NN      |       0.930465 |
| 20 | NN2     |           0.85137  |            | 20 | NN2     |       0.906483 |
| 21 | NN-CLF  |           0.920576 |            | 21 | NN-CLF  |       0.95276  |
+----+---------+--------------------+            +----+---------+----------------+
```

Table 12 Recall scores for validation and train datasets

The models that achieved the **highest recall scores** on the validation dataset are:

- NB: 0.993033
- B-Best: 0.965165
- DT-CP: 0.961449

Looking at the validation scores and the potential for overfitting, **the models above also show the best-balanced performance.**

It's important to note that the Naive Bayes Classifier (NB) has a high recall score most likely due to its inherent assumption of feature independence, which aligns well with the data distribution in this case. However, Naive Bayes overall performance on other metrics is relatively lower due to its simplistic assumptions.

F1-Score

The **F1 score** is a metric that combines both precision and recall, providing an overall assessment of the model's performance.

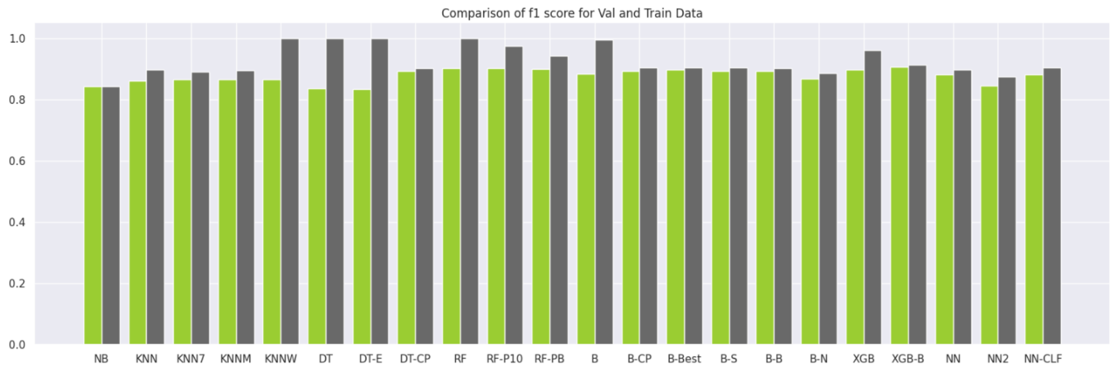
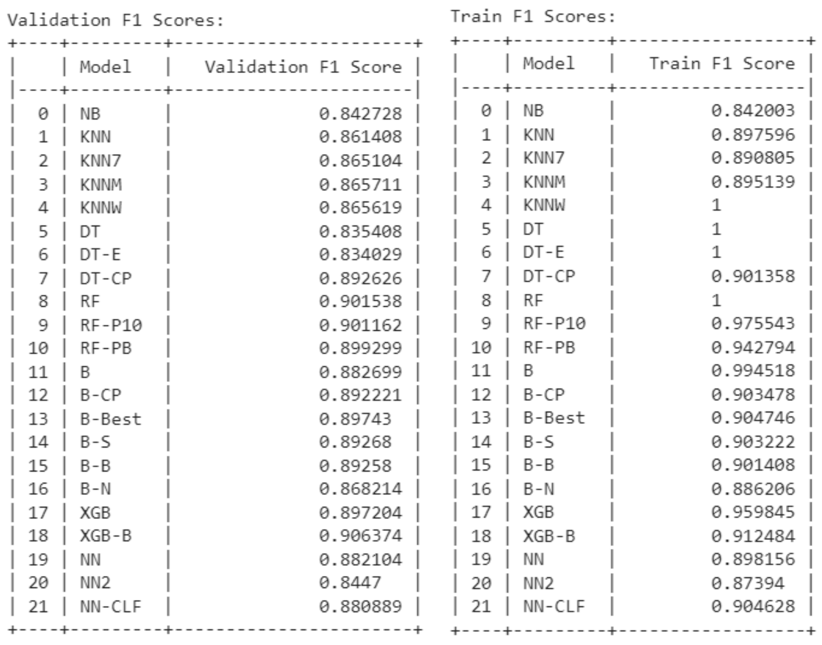


Figure 11 Comparison of F1 scores (green – val and grey - train)



```
Validation F1 Scores:
+----+---------+----------------------+
|    | Model   |  Validation F1 Score |
|----+---------+----------------------|
|  0 | NB      |             0.842728 |
|  1 | KNN     |             0.861408 |
|  2 | KNN7    |             0.865104 |
|  3 | KNNM    |             0.865711 |
|  4 | KNNW    |             0.865619 |
|  5 | DT      |             0.835408 |
|  6 | DT-E    |             0.834029 |
|  7 | DT-CP   |             0.892626 |
|  8 | RF      |             0.901538 |
|  9 | RF-P10  |             0.901162 |
| 10 | RF-PB   |             0.899299 |
| 11 | B       |             0.882699 |
| 12 | B-CP    |             0.892221 |
| 13 | B-Best  |             0.89743  |
| 14 | B-S     |             0.89268  |
| 15 | B-B     |             0.89258  |
| 16 | B-N     |             0.868214 |
| 17 | XGB     |             0.897204 |
| 18 | XGB-B   |             0.906374 |
| 19 | NN      |             0.882104 |
| 20 | NN2     |             0.8447   |
| 21 | NN-CLF  |             0.880889 |
+----+---------+----------------------+
```

```
Train F1 Scores:
+----+---------+-----------------+
|    | Model   |  Train F1 Score |
|----+---------+-----------------|
|  0 | NB      |        0.842003 |
|  1 | KNN     |        0.897596 |
|  2 | KNN7    |        0.890805 |
|  3 | KNNM    |        0.895139 |
|  4 | KNNW    |        1        |
|  5 | DT      |        1        |
|  6 | DT-E    |        1        |
|  7 | DT-CP   |        0.901358 |
|  8 | RF      |        1        |
|  9 | RF-P10  |        0.975543 |
| 10 | RF-PB   |        0.942794 |
| 11 | B       |        0.994518 |
| 12 | B-CP    |        0.903478 |
| 13 | B-Best  |        0.904746 |
| 14 | B-S     |        0.903222 |
| 15 | B-B     |        0.901408 |
| 16 | B-N     |        0.886206 |
| 17 | XGB     |        0.959845 |
| 18 | XGB-B   |        0.912484 |
| 19 | NN      |        0.898156 |
| 20 | NN2     |        0.87394  |
| 21 | NN-CLF  |        0.904628 |
+----+---------+-----------------+
```

Table 13 F1 scores for validation and train datasets

The models with the **highest validation F1 scores** on the validation dataset are:

- XGB-B: 0.906374
- RF: 0.901538
- RF-P10: 0.901162

After comparing the validation and train datasets scores to check for overfitting, **the model that achieved the best balance is still XGB-B**, since the high train scores of the other top performing models show signs of overfitting.

In conclusion, on the basis of the analysis of several metrics, the **XGB-B model** achieves the **highest scores in most metrics** and shows a lower **tendency to overfitting** compared to other models. As such, the **XGB-B model is the clear winner** in terms of performance evaluation.

8. Kaggle Submission

Finally, we made predictions on test data using our best model: **XGB-B**. Furthermore, we saved our predictions in a CSV file with only two columns: RecordID and our predictions, the *Outcome* variable. We achieved a score of **0.8686**.

During the semester, we made several submissions to Kaggle platform in order to obtain the best possible place in the competition. Over the semester we were constantly improving our preprocessing and optimizing our models so that we could achieve the highest F1 score value of the predictions.

9. Lessons Learned

This project was marked by key phases, which determined its progress and the results obtained.

The Pre-processing phase was undoubtedly the one that required the most time and dedication. We realized that investing time and effort into cleaning, transforming, and selecting relevant features from the dataset strongly impacted the quality of the results. In this sense, it was necessary to find the set of techniques that best suited the project dataset and that would allow, subsequently, to introduce the best and cleanest dataset for the training phase, consecutively reducing the GIGO (Garbage In Garbage Out) of our models.

Feature engineering also revealed to be very important. By selecting the most relevant features, we could significantly enhance the performance of our supervised learning models. Techniques such as feature scaling, target and ordinal encoding, helped us improve our models' predictive capabilities.

In turn, within each applied model, it was necessary to look for parameter adjustments that not only gave better results, but that, at the same time, prevented the model from having a very good performance on the training data but failed to generalize well to unseen or new data.

10. Conclusion

In conclusion, our objective in this project was to build a predictive model that accurately forecasts the performance of the athletes in their competitions. To ensure an organized and coherent workflow, we followed the CRISP-DM methodology.

After understanding the business requirements and exploring the dataset, we executed all the necessary steps to prepare the data for modeling. During the modeling phase, we built various algorithms and tested them aiming for the best score without overfitting. At the end, we made de predictions for our best model and submitted to Kaggle.

Self-reflecting on our work, we acknowledge that the preprocessing phase demanded the majority of our time and attention, since we were consistently trying to improve the data quality and make the necessary adjustments throughout the entire project development process.

Additionally, we found that the modeling phase was the one that we found most exciting. Experimenting different models and parameter combinations fueled our enthusiasm as we were always testing and aiming for the best score.

The Kaggle competition was also a motivating factor because we set ourselves the goal of finishing in first place.

Annex

XGBost

In this section, we aim to present a brief theoretical explanation of the extra algorithm and a justification for the chosen parameters.

Our group chose the XGBost (Extreme Gradient Boosting) as it is known for its effectiveness in handling various types of data and producing highly accurate predictions.

XGBoost is a powerful gradient boosting algorithm that iteratively trains a sequence of weak models (typically decision trees) to create a strong predictive model. By minimizing the sum of the loss function and a regularization term, this algorithm optimizes an objective function. The algorithm starts with a simple initial model and sequentially adds weak models that focus on the residuals of the previous models. The predictions of all the weak models are combined in a highly accurate ensemble model as a result of this repeated procedure. The effectiveness of XGBoost is attributed to its adaptability to different data types, regularization strategies to avoid overfitting, and efficient computation that scales well to huge datasets.

Due to our classification purpose, we implemented the XGBClassifier that uses the gradient boosting framework to create an ensemble of decision tree models. The XGBClassifier offers a set of parameters that can be used to optimize its performance. Therefore, the chosen parameters were carefully selected based on their impact on model performance. By setting appropriate values for max_depth, min_child_weight, gamma, subsample and colsample_bytree, we aimed to control model complexity, prevent overfitting, and fine-tune the model's performance for our dataset. This selection process was conducted by performing a grid search to find the optimal parameter values through cross-validation, as explained in section 6.4 Ensemble Classifiers - XGBost.

Support Vector Classifier

The Support Vector Classifier (SVC) is a machine learning model used for categorizing data. It finds the best possible boundary that separates distinct groups in the data. To address the problem that real-world data often requires more than a simple boundary for accurate separation, SVC uses a strategy known as Kernel Trick, specifically a type called Radial Basis Function (RBF), which gives more flexibility to handle complex patterns in the data.

In our project, we utilized SVC as the base estimator in a bagging technique. This approach involves creating multiple SVC models, each trained on different subsets of data, and then combining their outputs for a more robust prediction.

To enhance our model's performance, we fine-tuned several parameters within SVC. We used a loop to find the optimal number of estimators that would yield the best validation scores without overfitting the data.